AFIT/GCE/ENG/93D-02

GENETIC ALGORITHMS AND THEIR APPLICATION TO THE PROTEIN FOLDING PROBLEM

THESIS

Presented to the Faculty of the Graduate School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Engineering

Donald J. Brinkman, B.S.E.E, M.S.E.
Captain, USAF

December, 1993

ACKNOWLEDGMENTS

I would like to take this opportunity to thank everyone who played a part in this research effort. The largest thanks go to Larry Merkle, who always took the time to assist and guide my research efforts. His experience and insight was always greatly appreciated during times of frustration. I would also like to thank Dr Ruth Pachter who always encouraged me to keep up the good work, and the Materials Lab staff who were always willing to help with whatever support I needed. I appreciate all the effort and foresight which my advisor, Dr Gary Lamont provided throughout my thesis and course work. Although, he was often the cause of a great loss of sleep, he always seemed to make sure course work could some how be incorporated into my thesis research. Thus, nothing was ever a wasted effort, but rather proved to be very useful as graduation approached. I would like to thank all of my family, friends, and associates who have supported me throughout my college years. Finally, I would like to thank my wife, Lois, for her help, patience, understanding, and encouragement.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Abstract

The protein folding problem involves the prediction of the secondary and tertiary structure of a molecular system given the primary structure. The primary structure defines the sequence of amino-acid residues, while the secondary structure describes the local 3-dimensional arrangement of amino-acid residues within the molecule. The relative orientation of the secondary structural motifs, namely the tertiary structure, defines the shape of the entire biomolecule. The exact mechanism by which such a sequence of amino acids (protein) folds into its 3-dimensional conformation is unknown. Current approaches to the protein folding problem include calculus-based methods, systematic search, model building and symbolic methods, random methods such as Monte Carlo simulation and simulated annealing, distance geometry, and molecular dynamics.

Many of these current approaches search for conformations which minimize the potential energy of the molecule. Calculus-based techniques include steepest descent, conjugate-gradient, and Newton-Raphson methods to minimize a classical energy function of a molecular structure. The performance of calculus-based search, however, is susceptible to the many local minima which exist in such energy functions. Systematic search operates by enumerating all possible discretized conformations for a specified number of degrees of freedom. The exponential order of this technique makes it impractical for larger molecules where the number of degrees of freedom can range from a few hundred to thousands. Model building and symbolic approaches employ manual interaction to define and manipulate the initial 3-dimensional structure of a molecule and then a local optimization technique to refine it. Monte Carlo simulation and simulated annealing are stochastic semi-optimization techniques used to sample the search space for the lowest energy conformations. Distance geometry methods use an iterative process of

constraint satisfaction to reduce the upper and lower bounds of inter atomic distances within the molecule. The resulting tighter constraints allow a local minimization technique to find the final conformation. A genetic algorithm (GA), a stochastic search technique modeled after natural adaptive systems, potentially offers significant speedup over other search algorithms because of its inherent parallelizability.

The protein folding problem is represented in a GA as a population of strings which encode a possible conformation of the molecule being minimized. The GA uses an empirical calculation of the internal energy of the molecule as a fitness function to determine which population members survive into the next generation. Many parallel implementations of traditional search techniques distribute the calculation of the energy function across the processors which requires communication of the results to determine the overall energy. This workload distribution strategy often limits the scalability of the algorithm due to the fixed problem size of the energy function. However, a parallel GA can distribute the population members evenly across the processors. Each processor calculates the internal energy (fitness) of each of its population members independent of the other processors. Only minimal communications between processors is necessary in order to manage the overall population.

The results of applying a GA to the protein folding problem show significant improvement in execution time when compared to serial implementations of the GA. In addition, the parallel GA demonstrates good scalability characteristics since the communications strategy used to manage the population can be tailored to the parallel architecture.

GENETIC ALGORITHMS AND THEIR APPLICATION TO THE PROTEIN FOLDING PROBLEM

I. Introduction

As computers become faster and faster, the size of problems which are tackled using computers become larger and more complex. Computers of today are able to work with large amounts of data, manipulate this data, and solve difficult problems using sophisticated algorithms. Computer designers have, however, begun to approach the physical limitations of electronic components, effectively bounding the computational speed of single processors used to solve complex problems (DeCegama, 1989). Recently, designs for high performance computers have tended toward multiple processor systems. The goal of this new type of architecture is to have multiple processors working in parallel to speedup the overall throughput of the system. Ideally, the effect is a machine with an overall performance capability directly proportional to the sum of the performance of the processors which make up the system. However, the full potential speedup offered by parallel architectures is often never realized, primarily because it is difficult to simultaneously keep all processors busy doing useful work. Some overhead is associated with communications between processors which results in a drain on the performance of parallel architectures. To take advantage of the speedup offered by parallel architectures, algorithms must be designed to perform actions simultaneously, and with certain level of independence. Only when the algorithm is able to divide the workload up evenly among

all the processors, with minimal interprocessor dependencies, is the system able to achieve close to maximum performance.

However, even with the order of magnitude increases in hardware performance, there remain problems whose solution with ordinary algorithmic approaches remain unsolvable within a reasonable amount of time. These intractable problems grow exponentially, resulting in execution times on the order of years or centuries, even with the fastest hardware available. One such problem of international interest is called the protein folding problem.

The typical solution to solving intractable problems in a finite amount of time is to relax the solution constraint to accept near optimal or semi-optimal solutions rather than the guaranteed globally optimal solution (Cormen, 1990). Many search-based algorithms are available which provide good approximations to the global solution and offer reasonable execution times (Goldberg, 1989). However, these algorithms do not necessarily always yield good results. Often, they are problem specific and are consequently deceived by application problems outside of their intended problem domain.

Characteristics which can be used to evaluate algorithms include speed of execution, robustness of applications, and quality of solutions. To avoid unreasonable execution times, for problems of even moderate size, the order of complexity of the algorithm must not exceed low degree polynomial order. Often, exponential order programs are used despite the long execution times, simply because there is no currently available algorithm which can guarantee better results in less time. The limitation on this approach requires the user to be constantly aware of the algorithm's time complexity to avoid problem sizes which result in unacceptable execution times. This limitation often leads the user to ask, "What is a moderate size problem?" To help understand the impact of an exponentially increasing function, consider the following example. A computer capable of evaluating 10 trillion solutions per second would only be able to enumerate the

search space of a 2^n exponential problem up to a size of $n = 55$ before the execution time exceeds an hour. Solving the same problem for a size of $n = 60$ would take more than a day of constant execution, and for a problem of size $n = 70$, the computer would have to be left running for close to four years. It becomes obvious that for problems of relatively moderate size, a complete enumerative type algorithm is not a feasible option for solving exponential order problems. Consequently, advances in hardware alone will never achieve the desired speeds necessary to solve complex problems of large size, without corresponding improvements in algorithms.

1.1 Semi-Optimal Algorithms

A good algorithm should be designed not only to be of polynomial order, but should also be applicable to a variety of problems. A robust algorithm would allow a computer user to apply one algorithm to any of a variety of problems, and achieve good results. Algorithms tailored to specific problems usually have good or even excellent performance within their limited problem domain. However, these problem specific algorithms typically have poor performance outside of their intended application (Pearl, 1984). Consequently, computer science journals are filled with algorithms which have been shown to perform well for specific well defined tasks. Unfortunately, there is essentially a void of algorithms which offer acceptable performance across a wide spectrum of applications. Specifically tailored algorithms obtain their higher performance in terms of speed and solution quality by making use of previous knowledge of the problem domain to guide the search in the most efficient manner.

The quality of a solution is an important factor to semi-optimal search algorithms since by their nature, they do not enumerate the entire search space. Therefore, they can not guarantee the solution which was found is the globally optimal solution. Complete enumeration algorithms guarantee the globally optimal solution, but at the cost of

exploring, either explicitly or implicitly, the entire search space (Brassard, 1988). For exponential or combinatoric problems, exploring every element in the search space is simply not possible within a reasonable amount of time. The only alternative is to apply some strategy to limit the search space to a reasonable size which can then be explored.

1.2 Genetic Algorithms

Thus, it is desirable for an algorithm to be of polynomial time complexity, applicable to a wide variety of problems, achieve optimal or near optimal results, and also run efficiently on parallel machines. Genetic algorithms are a relatively new type of algorithm which have shown strong potential for meeting these objectives (Goldberg, 1989). Genetic algorithms are search based semi-optimization algorithms which are similar in some respects to random search based algorithms. The important difference, however, is genetic algorithms use a stochastic sampling procedure which results in a more directed search strategy than a purely random search. Genetic algorithms are of polynomial time complexity, and have been shown to achieve good results in a variety of applications. Modeled after the natural evolution process of selection, mating, and mutation, genetic algorithms simulate the Darwin theory of survival of the fittest. The search space is represented by a population of strings upon which genetic operators act to create new generations of strings. Starting with an initial population, the genetic operators use information about the fitness of the old population to create a new and better population with each successive generation. The solution to the problem corresponds to the string representation which possesses the highest fitness after an appropriate number of evolutionary cycles. Since populations can be easily divided into sub-populations, genetic algorithms are also inherently parallelizable.

As an example, of how a genetic algorithm could be used as a function optimizer, consider the following function, known as Rosenbrock's Saddle. The function is

characterized by a non-linear, non-convex shape which is difficult to optimize using a simple hill-climbing type algorithm.

$$f(x,y)=100(x^2-y)^2-(1-x)^2$$

In order to apply the genetic algorithm, the search space must be parameterized and encoded as a string. Consider a domain of the function from -2.00 to 2.00 with a desired resolution of 0.01 for each variable. This would require the search space to be discretized to at least 400 individual points per variable. Using a binary representation, an encoding of 9 bits is sufficient to represent 512 individual points. Therefore, the genetic string can be represented by a sequence of 18 binary characters, where the first half of the string represents the domain of variable x and the second half of the string represents the domain of y. Now, any point (x, y) can be represented in the genetic algorithm as a single 14 digit binary string, where the values of x and y are equal to the range of x and y times the quotient of the integer representation of the genetic string divided by the integer range of the genetic string minus the zero offset. For example, the values of x and y in Figure 1 are represented by the binary equivalent of 358 (256 + 64 + 32 + 4 + 2) and 234 (128 + 64 + 32 + 8 + 2). The corresponding real values are found by multiplying 4.0 (the range of x and y) by the quotient of 358 and 234 respectively, divided by 512, and then subtracting 2.0 for the zero offset.

The second step is to implement the means to evaluate the fitness of each member

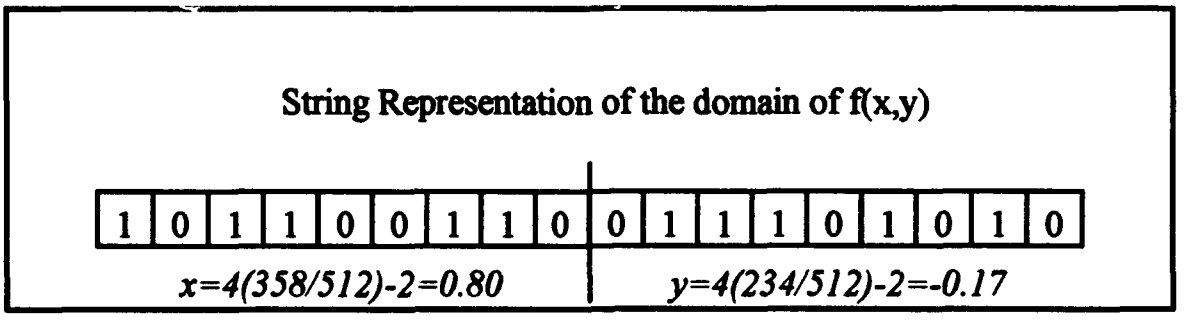

Figure 1. Example Encoding for a Genetic Algorithm String

of a population of strings which represent candidate solutions to the function being optimized. Assuming the function can be evaluated, the fitness value of each population member is simply the value of the function evaluated at the point (x, y) represented by the binary string. Through the searching action of the genetic algorithm, the population of strings evolves to conform to better and better solutions, eventually finding the optimal or near optimal solution to the function.

Despite all the advantages offered by genetic algorithms, they do have some limitations which lead to performance shortfalls when compared to more traditional search algorithms. Three primary deficiencies of genetic algorithms are premature convergence on local optima, lack of a stopping condition, and low precision (Eshelman, 1993; Palmer, 1991; Reynolds, 1990). These deficiencies are addressed in more detail in the literature review.

1.3 An Introduction to the Protein Folding Problem

The protein folding problem has the distinction of being among the national grand challenge problems. Simply stated, the protein folding problem consists of predicting the 3-dimensional structure of a protein, given the sequence of amino-acids which make up the protein. The implications of "solving" the protein folding problem are enormous. Knowing the 3-dimensional structure of a protein would allow biochemists to design new medicines with very specific properties, thus minimizing negative side effects. Engineers could use knowledge of the 3-dimensional structure of proteins to develop new materials designed with unique physical qualities. Finally, the genetic information encoded within our own DNA molecules may provide the answers to cure disease, birth defects, and other genetic abnormalities.

Currently, the only reliable methods available to determine the 3-dimensional structures of proteins is X-ray crystallography and Nuclear Magnetic Resonance (NMR).

However, both of these procedures require an extensive amount of lab time in order to determine the structure of one protein. The number of known protein sequences already outnumbers the number of known protein structures by more than two orders of magnitude (Lengauer, 1993). Given the current rate of discovery of new proteins, largely due to the work done in the Human Genome Project (US Congress, 1988), this gap is expected to continue to widen until a reliable and efficient method is found to accurately predict the 3-dimensional structure of a protein given the primary sequence of amino-acids which make up the protein. Even though a prediction algorithm is not as definitive as the actual determination of a protein's structure through NMR or X-ray crystallography, if it is able to demonstrate a reasonable level of accuracy at predicting the structures of already known protein sequences, then there will be a corresponding level of confidence in its ability to accurately predict the structures of new protein sequences.

The absolute solution to the protein folding problem can be modeled by two classical approaches (Lengauer, 1993). The first approach uses an *ab initio* energy minimization approach which accounts for all quantum-mechanical effects between atoms of the molecule; an $O(n^5)$ operation for each conformation. The final conformation corresponding to the minimum energy state, is then assumed to be the natural 3-dimensional structure of the protein. The problem then becomes, how to efficiently search the conformational space for the lowest energy structure. A complete enumeration of the search space is impossible, yet without explicitly or implicitly search the entire search space, the global minimum structure can not be guaranteed.

The second approach calculates the molecular forces acting on each atom of the molecule and applies Newtonian laws of motion to predict the trajectories of each atom as an N-body simulation. The greatest drawback to this approach is the required time steps, in order to accurately simulate motion at the atomic level, are much too small to be able to simulate the entire folding of a protein in our life time. Consequently, simplifications or

approximations necessary in order to increase the size of the time step. If the locations of each of the atoms near their globally minimum conformation could be approximated, then each of the classical approaches would be well suited to refine the final conformation. However, given an unknown sequence of amino-acids for which no structural information is available, a classical approach to structure prediction becomes intractable for even moderately sized molecules.

1.4 Parallel vs. Serial Computation

The primary obstacle to scaling to larger problem sizes for complex problems is the execution time usually exceeds an acceptable level. Several hours or even several day of computation time, may be an acceptable amount of time depending on the importance of the application. However, an execution time of several years is generally infeasible from both a cost perspective as well as a hardware reliability standpoint, no matter how important the application. As has already been stated, the processing speed of a single processor is currently approaching the physical limitations of the device. Consequently, a considerable effort is now being expended on parallelizing existing algorithms in the hopes of achieving speedup in execution times, thus allowing the algorithm to scale to larger problem sizes (Chandy, 1989).

Their inherent parallelizability of genetic algorithms is partially responsible for the increasing popularity. Several models have been developed to map the genetic algorithm to parallel architectures (Gordon, 1993). The particular model used depends largely on the type of parallel architecture. For distributed memory MIMD (multiple instruction multiple data) architectures, an island model usually works the best, where as a cellular model is considered to be the most appropriate for fine-grained, massively parallel architectures (Dorigo, 1993; Gordon, 1993).

1.5 Objective

The objective of this research effort is two fold. The first objective is to demonstrate that genetic algorithms can be successfully applied to the protein folding problem. The second objective is to demonstrate the scalability of a parallel implementation of a genetic algorithm as it is applied to the protein folding problem. The first objective involves developing an accurate model of the potential energy function of a molecule and incorporating as the fitness function of the genetic algorithm. The second objective requires the incorporation of the potential energy function into a parallel implementation of a genetic algorithm and executing the code on various numbers of processors to collect speedup information.

1.6 Assumptions

- The available AFIT software for simple and messy parallel genetic algorithms is assumed to function properly.
- Already existing performance data, which is used for comparative analysis is assumed to be obtained from algorithm implementations which have been optimized for maximum performance.

1.7 Scope

The scope of this project includes designing and implementing the empirical potential energy function of a molecule. The integration of this energy model as the fitness function of a sequential and a parallel implementation of a simple genetic algorithm is also covered. Comparisons are made with results from published studies of a simulated annealing approach. The problem domain consists of the multi-minima problem of finding the minimum energy structure of the polypeptide [Met]-Enkephalin. Application results are analyzed for both serial and parallel implementations of the genetic algorithm.

1.8 Standards

The following quantifiable measures are used as a minimum to compare the results of the implementations of the genetic algorithms, as well as results from non-genetic approaches to similar problems.

- Number of generations to find global optimum
- Execution time
- Error, difference between global optimum and solution found
- Space considerations

1.9 Approach/Methodology

1) Study literature pertaining to genetic algorithms, their applications, advantages, limitations, and current state of performance.
2) Study literature pertaining to the conformational analysis of proteins, including general background, genetic algorithm approaches, and other current approaches.
3) Comprehend and analyze the genetic algorithm software available.
4) Perform analysis and design of empirical potential energy function.
5) Perform analysis and design of molecular representation scheme to encode the conformation of a molecule as a binary string.
6) Implement empirical potential energy function, and integrate into existing code for simple and parallel simple genetic algorithms, using the sequential version of the genetic algorithm as a development environment.
7) Implement representation scheme for a molecular conformation.
8) Validate the potential energy model by comparing calculated energies with those obtained using the molecular simulation software package called CHARMm.

9) Test both the simple and parallel simple genetic algorithms against an energy minimization problem and analyze results to determine the tradeoffs between the two algorithms.

10) Compare results with published reports for simulated annealing approach and perform a quantified comparison between the performance of the two approaches.

11) Summarize results and discuss pros and cons of the two algorithms. Discuss the robustness of the algorithms with respect to the potential for scaling to larger parallel systems and larger problem sizes.

12) Discuss direction and recommendations for future work.

1.10 Materials and Equipment

All development is accomplished on AFIT workstations and the AFIT iPSC/2 Hypercube. Additional insight into the effect of parallelization of the algorithm is gained by porting the code over to larger systems, including the Intel iPSC/860 Hypercube computer in Beaverton, Oregon. Running the algorithms on various size machines provides an opportunity to observe the effects of scaling, and workload balancing of the algorithms.

1.11 Summary

Computer hardware is fast approaching physical limitations which bound the amount of computational power available from a single processor. Yet, computers are seen as essential tools to solve many problems of exponential complexity, such as the protein folding problem. For such problems, a completely guaranteed solution my not be possible, given their enormous complexity. Consequently, good approximation algorithms, applicable to a large variety of intractable problems, are necessary in order to reduce the overall complexity of finding at least a good solution. These algorithms should also be highly parallelizable in order to take advantage of the full speedup offered by

parallel processors. Genetic algorithms are viewed as a potentially promising technique for applications involving exponential order problems.

1.12 Layout of Thesis

This first chapter has already introduced the topics of genetic algorithms, the protein folding problem, and parallel processing. Chapter II is a literature review containing background information on genetic algorithms, current limitations of them, and finally, a review of current approaches to parallelizing genetic algorithms. Chapter III contains a summary of current approaches to conformational analysis, an analysis of the potential energy minimization approach to the protein folding problem, as well as a discussion of encoding schemes to represent the conformational search space of a protein. Chapter IV details the experimental design and implementation of the potential energy function, the representation of the problem, and the integration of the design into the genetic algorithm. Chapter V provides the results obtained from the genetic algorithm applied to the conformational analysis of the protein [Met]-Enkephalin. Finally, chapter VI concludes with some final observations and recommendations for future related efforts.

II. Literature Review of Genetic Algorithms

Although the concept of genetic algorithms is relatively new (Holland, 1975), there is a growing foundation of literature devoted to the theory of evolutionary algorithms and their application to a variety of problem domains. The first international conference on genetic algorithms was held in 1985, and has since contributed significantly toward the promotion of evolutionary computation. The proceedings of these conferences provide a wealth of information pertaining to both theory and application. The following sections provide a brief review of relevant background information, a discussion of current limitations, and a summary of various approaches to parallelize genetic algorithms.

2.1 Background

The primary distinction between various types of computer-based search algorithms is the manner in which the problem space is explored. Traditional search algorithms can be categorized as either calculus-based, enumerative, or random (Goldberg, 1989). Calculus-based algorithms seek to either indirectly derive the optimal solution by solving a set of equations, or directly find the maximum of the function by moving in the direction of the highest slope. An enumerative algorithm, which is only suitable for finite domains, simply enumerates or tests every point within the problem domain. A random search technique is similar to enumeration; however, only random points within the domain are tested. Genetic algorithms are search based semi-optimization algorithms which are similar in some respects to random search based algorithms. The important difference, however, is genetic algorithms use a stochastic sampling procedure which results in a more directed search strategy than a purely random

search. Genetic algorithms have been shown to achieve good results in a variety of applications including combinatorial minimization, such as the traveling salesman problem (Homaifar, 1993; Jog, 1989), the set covering problem (Sen, 1993), and functional optimization (Goldberg, 1987; Janikow, 1990; Muselli, 1992).

2.1.1 General Description. The basic genetic algorithm consists of at least three operators—*reproduction*, *crossover*, and *mutation*, modeled after the natural evolutionary processes of selection, mating, and mutation respectively (Goldberg, 1989; Holland, 1975). Genetic algorithms simulate the Darwin theory of survival of the fittest by representing the search space as a population of strings upon which genetic operators act to create new generations of strings. The application domain is encoded as a string structure representative of chromosomes. Through the application of selection operators to an initial population of string structures, a new generation is formed called *offspring*. Fitness values are calculated for each member of the population by applying a fitness function to each individual string of the population. The strings with the highest fitness values will be kept for *breeding* the next generation. The basic pseudo code for a genetic algorithm is presented below.

```
initialize population
calculate fitness for all members of the population
for i = 1 to max_number_of_generations
      for j = 1 to population_size
            crossover
            mutation
            evaluate fitness
      end loop
      reproduction
end loop
```

Reproduction employs a selection strategy to determine which strings will survive or be copied over into the next generation. The next generation then is made up of copies of strings with high fitness values which forms the mating pool for the following generation. The crossover operator mates a pair of strings by randomly selecting a

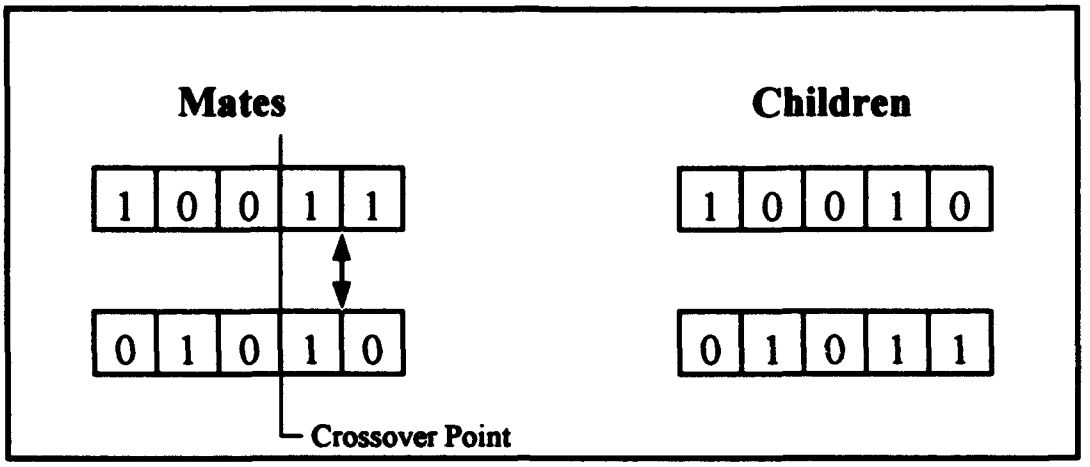

Figure 2. Example of Single Point Crossover

crossover point along the length of the string and swapping the string sequence from the crossover point to the end of the string (see Figure 2). The purpose of the crossover operator is to create new strings using pieces of the strings from the previous generation. The goal then is to combine the good parts of one string with the good parts of another string to form a new string which is better than each of the individual parents. The mutation operator randomly selects a string within the population and alters part of the string (see Figure 3). Just as in nature, the mutation operator is applied only periodically. By applying the genetic algorithm operators to an initial population and simulating the process of natural selection over many generations, a final population of the fittest strings is formed. The final solution found by the genetic algorithm corresponds to the string representation which possesses the highest fitness after an appropriate number of evolutionary cycles.

2.1.2 Complexity Analysis. The genetic algorithm is of polynomial order complexity with a finite space requirement determined by the population size. Although Goldberg (1989) suggests that there is an optimal population size which depends upon the length of the string, there is no fixed dependence between the length of the string and the

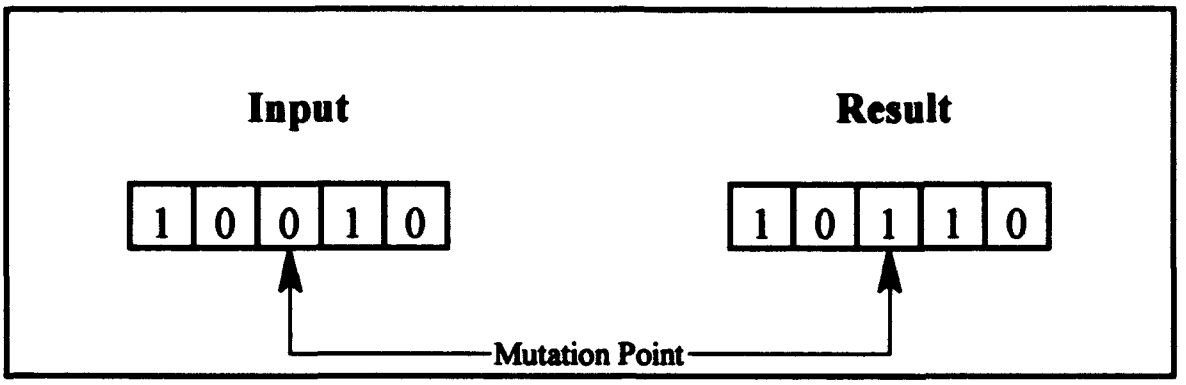

Figure 3. Example of a Mutation

population size. Experimental evidence, however, suggests that an insufficient population size may adversely affect solution quality (Goldberg, 1989 and Merkle, 1992). The terms in the order of the genetic algorithm reflect the length of the string as well as the number of strings in the population. An entire cycle of the genetic algorithm is executed up to a maximum number of generations specified by the user.

The various genetic operators each have associated minimum complexities, although the actual complexity depends on the implementation. The crossover operator selects two strings from the population pool of n strings, picks a random location along the length of the string of l bits, and then swaps the two tails of the parent strings which follow the randomly selected crossover point. The number of crossovers per generation is $O(n)$, and the actual crossover operation can be considered to be $O(l)$ since it needs to traverse the length of the string.

The fitness function includes a call to decode the string representation into the value in the problem domain. This decode call could be an $O(1)$ or an $O(l)$ function, depending upon the string representation scheme and the programming environment capabilities. Evaluating the fitness function is an $O(1)$ operation, with respect to the populations size, since it simply substitutes the decoded string values, and evaluates the

objective function. However, for complex problems, the evaluation of the fitness has a lower bound of the order of the objective function—the function being optimized.

Mutation also could be an $O(1)$ or an $O(n)$ operation depending upon the particular implementation and also the mutation strategy being used. Studies have shown good results are obtained with a mutation rate of once for every thousand string position transfers (Goldberg, 1989).

The selection function has an $O(n)$ complexity since it must evaluate each member of the population to determine which strings will be carried on to the next generation. Actual implementations of the selection operator may be of $O(n^2)$ complexity, such as a roulette wheel approach biased according to the fitness of the strings.

2.1.3 Niche Theory. While optimizing over a search space which contains many near optimal solutions, simple genetic algorithms have been found to be susceptible to genetic drift (De Jong, 1975; Goldberg & Segrest, 1987). Genetic drift happens when the stochastic error associated with the genetic operators, in conjunction with competing schemata corresponding to different peaks within the search space, causes the population to converge to one of the peaks (Deb, 1989). To overcome this pitfall, several implementations of genetic algorithms have been developed which attempt to maintain multiple, stable subpopulations on different peaks within the search space (Deb, 1989; Goldberg, 1989).

The theoretical foundation for maintaining separate subpopulations is based on the limited resources model which can be observed among natural organisms. In nature, organisms rarely compete for the same resource; rather, they coexist while exploiting different resources within the same environment. The relative size of the population of a species corresponds to the availability of the resource which it exploits. The balance of the number of individuals among any one species is maintained by the fact that all resources are limited; if a resource becomes over utilized, starvation will correct the

imbalance until an equilibrium is reached. The resulting system produces stable subpopulations of species which exploit their own region or niche within the overall environment.

The analogy between natural systems and genetic algorithms demonstrates the benefits gained by allowing the search space to be explored simultaneously within several different niches of the search space. The number of population members allocated to each niche is proportional to the relative fitness of the niche, corresponding to the availability of the resource. The balance of the number of individuals allocated to the various niches is maintained throughout the search process in two ways. The relative fitness of one niche can decline with respect to other niches which prove to yield better solutions, thus causing a proportionally smaller number of individuals to be allocated to the niche with lesser performance. The competition for limited resources also affects the probability of survival of any one individual. If a large portion of the population is exploiting one area of the search space, the survival rate of any one individual will be proportionally smaller, even though the area of exploitation may correspond to a niche of high fitness. This competition for limited resources provides the genetic algorithm with the mechanism to prevent the population from converging upon one solution.

*2.1.4 **Models and Niche Formation**.* There are basically two methods for incorporating a niche strategy in genetic algorithms. The first, proposed by De Jong (1975), is called the crowding scheme which implements a generalized form of preselection to determine which population member is replaced by a new offspring. The idea is to replace an individual with a similar individual of higher fitness, thus preserving population diversity. Similarity is based on a bit-by-bit comparison of two strings to determine the number of bits in common. The crowding scheme works by preventing members of one niche from dominating the members of another niche.

The second method used to create niches in the search space is called the sharing scheme (Deb, 1989; Goldberg & Richardson, 1987). The sharing scheme acts as a fitness scaling factor. The fitness of individuals are scaled according to the number of other similar individuals which considered to be exploiting the same region or niche within the search space. In this scheme a distance metric is used to scale the fitness of an individual such that its expected survival rate is proportional to the number and proximity of neighboring individuals. Consequently, when a large number of individuals inhabit the same region of the search space, their corresponding fitness will be reduced, resulting in a proportionally smaller number of individuals to be allocated to that region of the search space. Likewise, when an individual of relatively low fitness inhabits a sparsely populated region of the search space, its fitness will be scaled up, thus increasing it chances of survival.

The source of the distance metric used in the sharing scheme has lead to two different implementations. The distance used to quantify the proximity of neighboring individuals can be determined at either the phenotypic level or the genotypic level. The phenotypic level refers to the decoded parameter search space. For a problem with n parameters, the distance is usually calculated as the Euclidean distance between two points in n-dimensional space. The distance at the genotypic level simply corresponds to the Hamming distance between two strings of the population.

2.2 Limitations of Genetic Algorithms

Despite all the advantages offered by genetic algorithms, they do have some limitations which lead to performance shortfalls when compared to more traditional search algorithms such as A*, hill-climbing, depth-first, or breadth-first search (Rich, 1991). Three primary deficiencies of genetic algorithms are premature convergence on local optima, lack of a solution based stopping condition, and low precision. Some of these

deficiencies may be rectifiable through the use of dynamically self-adjusting genetic algorithms (Aizawa, 1993; Davis, 1989; Lee, 1993).

Currently there are implementations of genetic algorithms which incorporate, at least to some degree, a dynamic factor to adjust one or more of the evolutionary features of the genetic algorithm as the population evolves. The following sections focus on current implementations of genetic algorithms which incorporate a dynamic element within their genetic operators in order to adapt the search strategy to overcome obstacles during various stages of the evolutionary search process.

2.2.1 Premature Convergence. Premature convergence happens when a search through a search space becomes trapped in a locally optimal solution state and is unable to locate the globally optimal solution. This problem is particularly important for applications where many local minima exist, such that finding the globally optimal minimum becomes exceedingly difficult without exploring the entire search space. Although other algorithms are just as susceptible to this phenomenon, it may be possible to avoid premature convergence with genetic algorithms by modifying the way in which locally optimal states are explored.

In order for genetic algorithms to gain wider acceptance as a valid and reliable approach to solving real-world problems, efforts must be made to reduce the likelihood of premature convergence, and to increase the level of precision. The properties and characteristics of genetic algorithms are still not fully understood. Consequently, the exact causes of premature convergence are unknown. Studies have indicated evidence to suggest there may be a connection between population size and the probability of premature convergence (Merkle, 1992; Moed, 1991). Although these studies are not conclusive, and the data appears to be somewhat problem dependent, there is a direct relationship between population size and the tradeoff between exploration and exploitation (Moed, 1991). Convergence upon a local optima can have a profound impact upon

overall solution quality. A string of unusually high fitness, found early in the search process can quickly dominate the population as a result of the proportional allocation of the selection operator. In a parallel processing environment, migration strategies can also accelerate premature convergence by spreading locally optimal strings to other sub-populations (Merkle, 1992).

In an implementation by Reynolds (1990), the genetic algorithm was modified to include a dynamic fitness scaling factor. Scaling factors are often used to reduce the fitness of a string in order to promote the continuation of the search (Buckles, 1990, Reynolds, 1990). Reynolds justified the use of a scaling factor with the assumption that no single individual can represent the solution, rather the population must evolve to contain an entire generation of highly fit strings. LeGrand and Merz (1993) chose a different selection operator altogether, based on rank rather than fitness. This rank based approach is believed to reduce the problems associated with a string of unusually high fitness being assigned a disproportionate amount of the population space.

A variation of genetic algorithms, called messy genetic algorithms, have been developed in part to overcome problems of premature convergence (Goldberg, Korb, & Deb, 1989; Goldberg, 1990; Goldberg, 1991). The messy genetic algorithm is based on an assumption that the deceptiveness of the problem is bounded. A problem is considered deceptive if short, low-order building blocks lead to suboptimal higher order building blocks (Goldberg, 1989). One phase of the messy genetic algorithm, called the primordial phase, is designed to find the building blocks with the highest fitness values which will presumably be in the solution. The next phase, called the juxtapositional phase, evaluates the various combinations of building blocks to determine the combination which yields the highest solution quality. Messy genetic algorithms are more complicated than simple genetic algorithms, requiring both more time and more memory. Also, the primordial phase, which constitutes the majority of the execution time, is inherently non-parallelizable

on distributed memory architectures; however, approximations of the primordial phase do exist, which can be parallelized (Merkle, 1992).

*2.2.2 **Stopping Condition***. Users of genetic algorithms are forced to specify an artificial stopping condition corresponding to the desired number of generations to create. With each successive generation, the solution quality of the population evolves to higher and higher levels of fitness. There is, however, no obvious point at which the algorithm completes, or otherwise indicates that no further generations are necessary. Typically, the user specifies some arbitrary number of generations to perform, after which the final or best population found is accepted as the solution. It is difficult to estimate the number of generations required to evolve a solution of the desired quality; consequently, the number of generations is usually set much higher than the actual number needed. The trade-off which exists is between solution quality and execution time. An insufficient number of generations may result in a poor solution if it has not evolved long enough; excess generations waste computer time by performing evolutionary cycles which do not result in improved solutions. Since the algorithm is of polynomial order time complexity, the tradeoff usually favors wasting computer time to ensure an adequate number of evolutionary cycles are performed. However, even a conservative approximation based on previous observation does not guarantee an optimal stopping point.

There have been implementations of genetic algorithms which facilitate a simple automatic stopping condition which allows the algorithm to cycle through the evolutionary process, creating new generations until a threshold number of cycles have passed in which no new optimal solution has been found. The implementation by Reynolds (1990) contained an exit condition which terminated the genetic algorithm after a specified number of generations where all members of the population maintained a specified fitness level. In other words, as the algorithm proceeds, the overall fitness of the population increases, until a point where the population contains only strings of high fitness.

Furthermore, to maintain a high fitness value for a number of generations, any recombination's of strings within the population must also be of a high fitness. Assuming the best solution found so far is being stored and updated throughout the genetic algorithm, a simple way to facilitate an automatic stopping condition is to allow the algorithm to cycle through the evolutionary process, creating new generations until a threshold number of cycles have passed in which no new optimal solution has been found. LeGrand and Merz (1993) used a multiple exit criteria to terminate the search when any of the following conditions were met: 100,000 generations without finding a new solution to replace the previously best solution, variance of the population fitness falls below 0.1, or the distance between 200 randomly selected pairs falls below a specified amount. The second and third of these exit criteria, like the strategy used by Reynolds, exploits some measure of the population convergence to generate a terminating condition.

*2.2.3 **Precision***. A final limitation of genetic algorithms is the lack of precision in exploiting a minimum state. Often, a genetic algorithm is able to locate the general location of a local or global minimum, but the nature of the genetic operators make it difficult to narrow in on the specific minimum of the surface or function (Janikow, 1991). A typical approach to this problem is to use genetic algorithms to locate the vicinity of minimum states to limit the search space and then apply a different algorithm, such as a gradient based algorithm, to perform the fine optimization.

To improve precision, genetic algorithms have been used in conjunction with gradient-based algorithms to maintain the robustness of genetic algorithms, yet gain the fine precision of a directed search algorithm. A study by Janikow and Michalewicz (1990) implemented a specialized genetic algorithm able to achieve precision by combining a floating point representation scheme with what was referred to as a dynamic mutation rate, which decreased with time. The effect of the adjustable mutation rate was a more localized search with each successive generation. Janikow and Michalewicz argued a

decreasing rate of mutation causes the population to be searched very locally at later stages of the evolutionary process. Similarly, Atmar (1990) claimed decreasing the mutation rate stabilized the genetic information, thus avoiding the disruptive nature of mutation on an already highly fit string. The results of experiments performed by Janikow and Michalewicz show their specialized genetic algorithm outperforms classical approaches in terms of both precision and the required number of generations. They also found the degree of precision achieved by their implementation compared favorably to more traditional gradient-based approaches.

2.3 Parallel Genetic Algorithms

There are a number of strategies for mapping of genetic algorithms to distributed memory architectures. Selection of a particular approach should take into consideration certain features of the target architecture:

- the time t_{comm} required for a fixed amount of interprocessor communication,
- the computation time t_c required for a single fitness function evaluation, and
- the number of available processors.

Of particular importance is the ratio t_c/t_{comm}. Using this ratio, it is possible to determine the amount of interprocessor communication which may be performed without sacrificing processor utilization. Typically, $t_{comm} >> t_c$, implying that very little interprocessor communication may be performed without impacting execution time.

One strategy, which has very flexible interprocessor communication requirements, is the "island" model, wherein the population is partitioned into subpopulations (Dorigo, 1993; Gordon, 1993). Each processor executes an independent genetic algorithm operating on a single subpopulation. No interprocessor communication is required for fitness evaluation. Various global selection and/or migration strategies may be employed.

Experimental results indicate that tradeoffs exist between solution quality and required communication time.

A less common strategy is the "farming" model, in which the whole population resides on a single processor (Dorigo, 1993). This processor (the *master*) applies the genetic operators to the population, while the remaining processors (the *slaves*) perform the fitness function evaluations. As the master processor creates a new individual, it requests one of the slave processors to evaluate its fitness. After evaluating the fitness of the population member, the slave returns the fitness to the master. Since $O(n)$ communications are necessary at every generation, this strategy is best suited for applications in which $t_c \approx t_{comm}$.

Recently, many parallelized versions of several molecular dynamics programs, including CHARMm, have become available. This offers many different possibilities for decomposing a GA approach to the structure prediction problem. For example, a single global population could be executed on the host where the evaluation of fitness for each population member is executed in parallel on the node processors. This would in effect speed up the evaluation cycle of the GA. However, one must consider the added communications required to communicate each population member to the node processors for every generation.

Parallel implementations of genetic algorithms provide a perfect setting for encouraging the formation of niches within the search space. The subpopulations present on each node of a multiprocessor environment provide a natural mechanism to implicitly search separate niches of the search space, as opposed to explicitly forcing the formation of niches within a single population in a serial genetic algorithm implementation. However, some communication may be necessary between nodes in order to prevent a significant amount of duplication between subpopulations.

2.4 Summary

Although it has been almost twenty years since genetic algorithms were first established by John Holland (1975) as a valid technique for searching complex spaces, it has not been until recently that attention has focused on using genetic algorithms as general problem solvers for a wide variety of applications. Already genetic algorithms have been shown to achieve good solutions for solving complex business, science, and engineering problems. As the variety of applications increases, solving some of the current limitations, including premature convergence, termination, and precision, becomes all the more important. Previously, limitations in the performance of genetic algorithms in a specific application could be compensated by the integration of domain specific knowledge. However, to allow genetic algorithms to be used as general purpose tools, the strategies used in the evolution process must be independent of the application domain.

The parallelization of genetic algorithms potentially offers an attractive means for achieving near linear speedup using parallel processing. The ability to tailor the communication strategy of a genetic algorithm to the particular parallel architecture, allows the time penalty of communications to be minimized. Consequently, the most natural method of parallel decomposition of the genetic algorithm is to distribute the total population as a set of subpopulations on each node, where the fitness of each member of the subpopulation is calculated locally. It is the independence of the subpopulations which allows a parallel GA to achieve near linear speedup. Even though the evaluation of the fitness may be parallelizable, it is unlikely that the interprocessor communications required to perform an evaluation in parallel could be less than the communications between subpopulations.

As the present understanding of genetic algorithms grows, knowledge of the effects of genetic operators on solution quality and performance will allow better search

strategies to be developed. Currently, dynamic operators hold promise for improving the performance of genetic algorithms by adjusting their influence to reflect the stage of the evolutionary process. A dynamic selection process may reduce the likelihood of premature convergence on a locally optimal solution by preventing strings of unusually high fitness from dominating the population early in the search process. A dynamic termination condition may eliminate the need to specify an arbitrary number of generations by gauging the level of convergence of the population. A dynamic mutation rate may allow more precise solutions by reducing their disruptive influence during the final optimization stages of the search. Perhaps the biggest benefit of implementing dynamic operators is that the algorithm may be able to adjust itself to a wide variety of problem domains, without the intervention of a domain expert.

III. Analysis of the Protein Folding Problem

A protein consists of a sequence of amino acids. Each amino acid is identified by an attached chemical structure called a sidechain (LeGrand, 1993). The common portion of an amino acid, to which the side chain or residue is attached, forms the backbone of the protein molecule (see Figure 4). The sequence of amino acids, linked together by peptide bonds, forms what is referred to as the primary structure of a protein. Atomic forces and interactions between bonded atoms cause the primary structure to fold into a three dimensional structure, known as the tertiary structure. Contained within the tertiary structure are distinguishable structural components, such as alpha helices or beta sheets, which are called secondary structures. The final tertiary structure, where all bonded and non-bonded interactions are stable, is called the molecular conformation. The protein folding problem can be described as a search for the natural tertiary structure of a protein, given its primary structure.

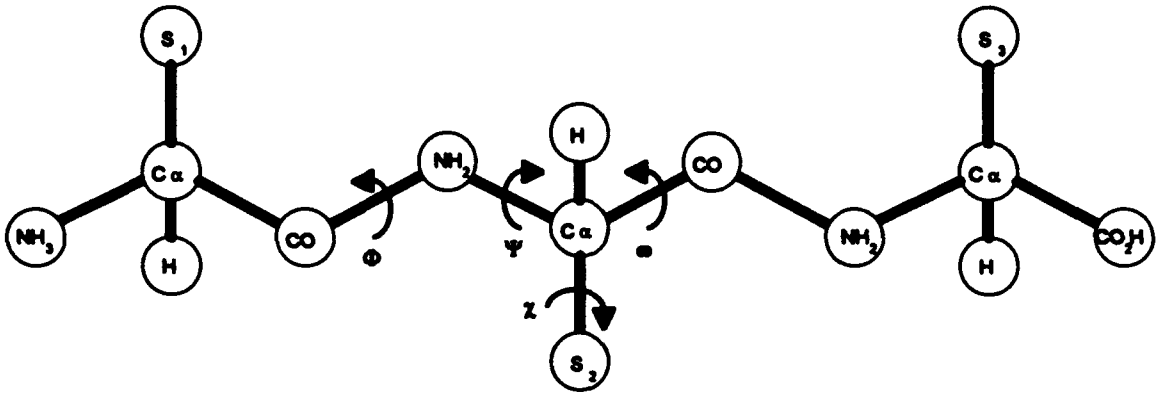

Figure 4. A Three Amino Acid ProteinA

3.1 Background

Proteins are an integral part of everyday life. The function of any protein is largely determined by its tertiary structure (Chan, 1993; Lengauer, 1993). Fibrous proteins are found in structural components of our bodies, while membrane proteins mediate the exchange of materials across cellular boundaries. Enzymes, which control virtually all biochemical reactions in living cells, are globular proteins. A considerable amount of research today is directed toward biochemically engineering proteins with desired functional properties. However, in order to accurately predict the functional properties of a protein, some knowledge of the tertiary structure is required. The development of an accurate model for predicting the tertiary structures of known protein sequences would significantly aid biochemists in their search for proteins with specific functional properties.

Current technology provides a virtually automated means of sequencing a protein to determine its primary structure. Consequently, there are about 50,000 known protein sequences (Bairoch & Boeckmann, 1991). Conversely, current technology such as X-ray crystallography and nuclear-magnetic-resonance (NMR), has only been able to experimentally determine the tertiary structure of about 400 proteins (Bernstein, et. al, 1977). This gap in structural information is expanding due to the Human Genome Project which at its current rate is doubling the number of known protein sequences every year (US Congress, No. OTA-BA-373, 1988).

Another indication of the scale of the protein folding problem is the complexity of the conformational search space. The algorithmic complexity of an optimization problem depends on two factors—the number of variables in the function to be optimized and the size of the variable domains. The size of the search space is proportional to the variable domain, but can also be scaled by a factor associated with the desired precision of the solution. To analyze a problem in terms of computational complexity, the solution space must be viewed as a discrete representation, since any computer driven solution must

represent the search space as a discrete set of points. The overall complexity of finding the globally optimal solution to a problem then is the time associated with implicitly or explicitly searching the entire solution space. For complex, non-linear problems, complete enumeration is the only guaranteed method for determining the globally optimal solution. Thus, the complexity of a problem containing n variables denoted by $x_1, x_2,\ldots,x_n$ would be the following product term.

$$\prod_{i=1}^{n} \|d_i\| \quad \text{(where } \|d_i\| \text{ is the cardinality of the discrete domain for the variable } x_i\text{)}$$

There exists a relationship between the number of atoms in a molecule and the degrees of freedom associated with the molecular energy. Each molecule has $3n$ - 6 degrees of freedom where n is the number of atoms contained in the molecule. This relationship simply results from assigning the location of each atoms with respect to one of the molecule's atoms chosen as the reference point. Each protein consists of anywhere from a few to several thousand amino acids, and each amino acid contains a number of atoms. The resulting search space in which a conformation can be found is a hyperspace of dimensionality $3n$ - 6.

The exponential order of the search space grows much faster than 2^n or 3^n since each dimension of the search space can be discretized to some domain of possible values d resulting in a complexity of $\|d\|^{3n-6}$ search space. Fortunately, the bond lengths and bond angles of a molecule are relatively stable. The principal determinants of the tertiary structure of a protein are the dihedral angles of the molecule (LeGrand & Merz, 1993). Consequently the conformational search space of a protein can be reduced to $\|d\|^n$, where $\|d\|$ is the number of discrete dihedral angles represented, and n is the number of independently variable dihedral angles. However, even for this simplified problem, a complete enumeration of the total conformational search space is infeasible. For example, consider a small problem with only 15 independently variable dihedral angles and a range

of dihedral angles from 0 to 360 degrees discretized to 20 degree increments. The resulting search space is of the order of 10^{18} possible conformations. Even using a teraflop computer which evaluates one point every clock cycle, a complete enumeration technique would require over 78 days to search this relatively small search space. Adding just one more independent variable increases the execution time to almost 4 years. The complexity of the problem, combined with the ever increasing rate at which protein sequences are being discovered, drives the need for faster methods of determining the tertiary structures of known protein sequences. This is the reason the protein folding problem is included as one of the national grand challenges.

3.2 Current Methods of Structure Prediction

There are currently several approaches to finding the natural, three dimensional conformation of proteins. Methods of conformational analysis include molecular dynamics, energy minimization, homology-based, and simplification techniques.

*3.2.1 **Molecular Dynamics**.* Perhaps the most natural approach is to actually simulate the folding process. This is the approach taken when using molecular dynamics to model the motion of atoms as an N-body simulation, using Newtonian laws of motion in response to atomic forces. The limiting factor to the use of molecular dynamics is the time step required to accurately model the motion of atoms as the protein folds. Thermal oscillations within a molecule require the simulation time step to be on the order of femtoseconds (10^{-15}). However, the complete folding process is on the order of milliseconds or even seconds (Lengauer, 1993). Current technology limits the simulation of atomic trajectories to time windows of only a few hundred picoseconds (10^{-12}); much too short to determine the final folded tertiary structure.

*3.2.2 **Energy Minimization**.* Energy minimization methods are based on the observation that physical systems tend toward a minimum energy state. Whether the

natural conformation of a protein necessarily coincides with the global minimum energy state of the molecule is still undetermined. Nonetheless, the natural conformation of a protein must coincide with at least a locally minimum energy state in order to be stable. Thus, the search for the natural conformation equates to a functional optimization problem where the function to be minimized is a model of the energy of the protein with a specific three dimensional structure. The are two major obstacles which energy minimization techniques must face. The first obstacle is developing an accurate model of the energy function. An exact model, accounting for all quantum-mechanical effects can be an expensive calculation with a time complexity as high as $O(n^5)$, where n is the number of atoms in the molecule (Lengauer, 1993). Consequently, semi-empirical calculations are usually used to approximate the energy function as an $O(n^2)$ operation. The second major obstacle is to devise a search algorithm which efficiently samples the conformation search space, yet avoids the multi-minima problem of becoming trapped in local minima.

There are many approaches to functional optimization (Bilbro, 1991). Differential algorithms use brute force mathematics to derive the minimum of the function. This approach, however, only works when the function is differentiable. For non-differentiable functions, gradient-based or hill-climbing algorithms can be used. A gradient-based approach uses a greedy type algorithm to direct the search in the most promising direction. The greedy algorithm operates on the basis of local decisions to guide the search toward the globally optimal solution. This approach works fine for simpler functions, but does not perform as well on complex functions containing many minima. Consequently, a more robust search strategy must be applied to avoid being trapped by local minima.

The search space for a function consists of the domain of the variables contained in the function to be optimized. The solution will be the set of n values, where n is the

number of variables in the function, and $f(v_1, v_2, v_3, ..., v_n)$ is either a maximum or minimum solution depending on the type of optimization being performed.

$$\text{Solution of } f(x_1, x_2, x_3, ..., x_n) = [v_1, v_2, v_3, ..., v_n]$$

The solution space of the energy function contains many local minima, and is a fundamental challenge to functional optimization techniques to find the globally optimal solution. Various search algorithms, including Monte Carlo optimization, simulated annealing, and genetic algorithms, have been used to address the multi-minima problem.

*3.2.3 **Homology-Based Approaches.*** A different type of approach altogether, uses a more qualitative foundation. The use of homology to predict the structure of proteins is based on the premise that molecules with similar primary structures will form similar tertiary structures (Lengauer, 1993). Therefore, if the amino acid sequence of a new protein is found to be similar to that of a protein with known tertiary structure, then based on analogy, the new protein will possess a similar tertiary structure. The obvious limitation of this methodology is its inability to extrapolate to new proteins which differ significantly in their primary structure from any known protein. A variation on this theme focuses on the secondary structures which make up the overall tertiary structure. The problem reduces to aligning sequences of a new protein to the sequences of a protein with known tertiary structure. These sequences are then assumed to correspond to the secondary structures contained within the known protein. Although this approach allows the extrapolation to conformations of unique and different proteins, it ignores the interactions between secondary structures and their effects on the overall tertiary structure. Also, the sequence alignment problem is a combinatoric problem in itself to find the best match between two amino acid sequences.

*3.2.4 **Simplification techniques.*** Simplifying assumptions have often been used to model the complex behavior of large problems in a feasible and representative manner.

The idea behind simplification is to capture the gross contributions to a complex problem, and simplify the rest so as to account for the majority of the problems behavior (Lengauer, 1993). A common simplification technique used in conformational analysis is to restrict the conformation of a protein to a lattice type structure where the backbone-carbon atoms (C-alpha's) are forced to reside on the vertices of the lattice. The connections between vertices of the lattice are chosen such that they represent likely conformation angles, thus greatly reducing the conformational search space. Lattice models usually simplify or even omit the modeling of the side chains of the amino acids. Despite their many simplifying assumptions, lattice models have been found to exhibit protein-like behavior when modeling hydrophilic and hydrophobic behavior. Hydrophilic proteins are polar or charged and are attracted to water, where as hydrophobic proteins are oil-like and repel water (Chan, 1993).

3.3 The Semi-Empirical Potential Energy Function

The foundation for using a semi-empirical potential energy function is based on a tradeoff between numerical accuracy and calculation complexity. The semi-empirical potential energy function presented here, is derived from extensive experimental analysis (CHARMm, 1992; Le Grand, 1993). The parameters for the constant terms of the expression are also experimentally determined. The function takes into account the non-bonded van der Waals interaction represented by the Lennard-Jones potential, Coulomb's law, and the interactions between bonded atoms. Note that for non-bonded electrostatic interaction, the solvent interaction between the atoms of a molecule and the surrounding environment are not included explicitly.

The molecular energy equation is a non-linear function, whose parameters are difficult to optimize due to the complex interactions taking place between the atoms of a molecule which the function models. The simplest term of the energy function includes

only the non-bonded interactions term of the equation. The function defined below is the Lennard-Jones potential which accounts for the van der Waals attraction and repulsion energy. The last term in the non-bonded interactions function models the electrostatic attraction and repulsion energy.

$$E_{non-bonded} = \sum_{non-bonded} \left[\left(\frac{A_{ij}}{r_{ij}} \right)^{12} - \left(\frac{B_{ij}}{r_{ij}} \right)^{6} + \frac{q_i q_j}{\varepsilon r_{ij}} \right]$$

The terms A_{ij} and B_{ij} are empirically determined constants for atoms i and j, and r_{ij} is the distance between atoms i and j measured in Angstroms. The q_i and q_j terms represent the atomic charges of atoms i and j. The total energy from non-bonded interactions is defined as a summation over all pairs of atoms within the molecule. For only two atoms, this function has a very deterministic behavior, but as the molecule of interest becomes more complex, the number of atomic interactions increases polynomially with each additional atom added to the molecule.

The next step is to add the components of the energy function which represent the interactions along molecular bonds. This energy is represented by three summation terms which vary over the bond lengths, bond angles, and the dihedral angles formed by the molecular bonds. Empirically derived constants are used in the equation and consist of a leading constant coefficient (K) associated with the types of atoms involved in the relationship. An equilibrium value (designated by *eq*) for bond length, bond angle and dihedral angle is also present and depends upon the types of atoms in the relationship. The individual terms of the energy function are described below.

$$\text{Bond length energy} = \sum_{bonds} K_{r_{ij}} (r_{ij} - r_{eq})^2$$

$$\text{Bond angle energy} = \sum_{angles} K_{\Theta_{ijk}} (\Theta_{ijk} - \Theta_{eq})^2$$

$$\text{Dihedral angle energy} = \sum_{dihedrals} V_{\Phi ijkl}\,[1 + \cos(\Phi - \gamma_{ijkl})]$$

The total energy of the molecule is equal to the summation of the individual terms which define the bonded interactions plus the summation of the non-bonded interactions terms. The total potential energy of a molecule is defined as follows.

$$E_{Total} = \sum_{bonds} K_{r_{ij}}\,(r_{ij} - r_{eq})^2 +$$

$$\sum_{angles} K_{\Theta_{ijk}}\,(\Theta_{ijk} - \Theta_{eq})^2 +$$

$$\sum_{dihedrals} V_{\Phi ijkl}\,[1 + \cos(\Phi - \gamma_{ijkl})] +$$

$$\sum_{non-bonded}\left[\left(\frac{A_{ij}}{r_{ij}}\right)^{12} - \left(\frac{B_{ij}}{r_{ij}}\right)^{6} + \frac{q_i q_j}{\varepsilon r_{ij}}\right]$$

3.4 Encoding of the Search Space

As stated previously, the number of independent variables of the energy function is equal to $3n$-6, where n is the number of atoms in the molecule. These independent variables correspond to the bond lengths between bonded pairs of atoms, the bond angles formed by two pairs of bonded atoms with a common atom forming the vertex of the angle, and the dihedral or torsion angle defined as the rotation angle along the axial center (bond) of a chain of four bonded atoms. The bond lengths and bond angles within a molecule exhibit relatively stable behavior and consequently can be accurately modeled as a fixed variable based on the atom types involved in the bonded interaction (Schulze-Kremer, 1993). Tables are available which provide empirically determined values for bond lengths and bond angles for most combinations of atoms. The dihedral angles constitute the remaining independent variables. To further refine the problem, attention can be

focused on the phi, psi, and omega (Φ, Ψ, and ω) angles (dihedrals along the backbone) of the protein molecule, and the chi (χ) angles of the residue (dihedrals determining the orientation of the sidechains). The remaining dihedral angles contained within the chemical structures of the sidechains are held fixed.

The solution space for the potential energy function is an *n*-dimensional hyperspace where *n* is the number of independently variable dihedral angles. The surface of the energy function is characterized by many local minima, resulting in a function which is difficult to optimize (Bilbro & Snyder, 1991). A simple gradient based search algorithm would likely become trapped in a local minima, unable to proceed to find the globally optimal solution. Stochastic based search algorithms such as Monte Carlo, simulated annealing, and genetic algorithms have been shown to successfully avoid the multi-minima problem of becoming trapped in a local minima. Each of these search techniques are able to use the potential energy function to evaluate points in the search space corresponding to a specific conformation of the molecular structure being minimized. The search space consists of an encoding of the independent variables necessary to fully specify the three dimensional structure of the protein.

3.4.1 A Genetic Algorithm Encoding Scheme. A method which can be used to search for the maximum or minimum of a complex function, using genetic algorithms starts with constructing an encoding of the search space as a character string. This involves two decisions, the cardinality of the alphabet used as the characters, *c*, and the length of the string, *l*. Both of these decisions will affect the overall size of the search space since the size of the search space is defined by c^l. For functional optimization, the most common string representation uses a binary alphabet to define a string of length *l*, where *l* is determined by the size of the largest possible solution for each of the variables in the function to be optimized. For example, a function of two integer variables with a domain of 0 to 100 could be represented by a string of length 14, the first seven of which

would represent the binary encoding of one variable, and the second half of the string would represent the other variable. Although this string would actually represent a possible solution space of 0 to 127 for both of the variables, the simplicity of the string representation, and the relatively small string length make this the most obvious and practical choice of string length and alphabet. However, as the size and complexity of the function to be optimized increases, a more efficient means of representing the search space may be necessary.

The grain of the search space is another consideration which may influence the effectiveness of a representation scheme. For example, if a function contains trigonometric components, it may only be feasible for solutions to exist on increments of π radians, even though the function may be defined for all real numbers. The grain of the string representation is very specific to the problem, and may require a significant understanding of the problem in order to reduce the grain of the representation. The benefit, however, of reducing the granularity of the encoded string, is a much smaller search space.

Often, the binary alphabet is chosen since the genetic operators necessary for crossover and mutation are easy to implement for binary strings. Also, the conversion procedure to convert the encoded representation into the actual solution format in order to evaluate the objective function should also be efficient since it is executed often. Experiments by Janikow (1990), using a floating point string representation, indicate that non-binary strings are also capable of providing good results.

IV Experimental Design and Implementation

The design of any experiment requires a balance between planning and a constant awareness of the goal of the research effort. The most carefully laid out experiment is meaningless unless it is able to substantiate the fundamental hypothesis of the experiment. Consequently, an experiment should be designed to provide some quantifiable measure from which to perform statistical analysis and draw conclusions (Montgomery, 1976).

The following section discusses the details of the experimental design; the implementation and validation of the molecular energy model, the integration of the model into existing software, and the program execution on various hardware architectures. The section concludes with a detailed presentation of the methodology used to evaluate the performance of this GA approach to the conformational analysis of proteins.

The approach used to experimentally evaluate the performance of GA's for conformational analysis involves a three step process. The first step is to develop an accurate model of the molecular energy. The next step is to integrate the model into the GA. Finally, the last step is to design an experiment which evaluates the performance of GA's over a variety of test cases.

4.1 Implementation of Molecular Energy Model

The molecular energy function is an empirical calculation of the molecule's energy based on the geometry between various sets of atoms within the molecule. Consequently, a data structure which captures the three dimensional structure of the molecule is required in order to evaluate this function. The empirical energy function must also have access to

constant parameters which are determined by the types of atoms involved in the energy term. Finally, a procedure must be devised to validate the energy model.

4.1.1 Initialization of Data Structures. A systems approach to design starts with an analysis of the inputs and outputs of a process to determine its requirements. The input files used in this research are generated by a software package called QUANTA. The primary input to the energy model of a particular protein is the Z-matrix representation of the protein. A Z-matrix encodes the structural information necessary to fully define the 3-dimensional position of each atom within the molecule. The method in which this structural information is encoded is by defining each atom's position in terms of its bond length, bond angle, and dihedral angle with respect to three previously specified atoms. Generating a Z-matrix with QUANTA produces an output file with the following format for each line of the file.

[atom type] [bond length] [flag] [bond angle] [flag] [dihedral] [flag] [atom_j] [atom_k] [atom_l] [charge]

Z-matrix Format

The Z-matrix consists of a sequential listing of all the atoms present in the molecule. Each line of the Z-matrix corresponds to a specific atom number. The atom type is simply the chemical symbol for the type of atom present. The bond length is the distance between the present atom and the atom corresponding to the number in the field atom_j. The bond angle is defined as the angle formed between the present atom, atom_j, and atom_k. The dihedral angle is the twist or torsion angle along the central bond of a chain of four atoms. Thus, the dihedral angle is defined as the torsion angle along the axis of the middle bond formed between the present atom, atom_j, atom_k, and atom_l. The flag fields present in the Z-matrix serve the purpose of identifying structural parameters as either fixed or independently variable. Although the Z-matrix contains a field for the

charge of each atom, this field is not used. Instead, QUANTA generates a separate file, an rtf file, which contains more specific atom type information as well as the atomic charges corresponding to each atom of the molecule.

N	0.00000	0	0.00000	0	0.00000	0	0	0	0	0.000
C	1.45300	0	0.00000	0	0.00000	0	1	0	0	0.000
C	1.52877	0	107.59007	0	0.00000	0	2	1	0	0.000
C	1.49994	0	111.60606	0	-119.97103	1	3	2	1	0.000
C	1.38714	0	120.03769	0	89.96887	1	4	3	2	0.000

Example lines from a Z-matrix

The above example shows the first five lines extracted from a Z-matrix for [Met]-Enkephalin. From this small example the following information can be interpreted: the fifth atom is a carbon atom, which is 1.38714 Angstroms distant from the fourth atom of the molecule, and forms a 120.03769 degree angle between itself, the fourth and the third atom, where the fourth atom lies on the vertex of the angle. Likewise, there is a torsion angle of 89.96887 along the bond between the fourth and third atom with respect to the chain of atoms extending from the fifth to the second atom. A "1" in the flag field is used to indicate an independently variable parameter, while a "0" indicates that the parameter is held fixed. The flags in the example indicate the dihedral angles for the fourth and fifth atoms are independent variables.

In three dimensional Cartesian space, the first atom of the Z-matrix may be arbitrarily considered to be at the origin. The second atom may be considered to be on a coordinate axis, such that the bond between the first and second atoms lies on either the x, y, or z axis. The position of the third atom may be considered such that the plane defined by two unit vectors along the bonds to the two previous atoms lies in either the xy, yz, or the xz plane. The Cartesian coordinates of the fourth atom, as well as any subsequent atom, can then be calculated given the distance, bond angle, and dihedral angle between that atom and the three previously specified atomsto which it is bonded. Thus, the process of calculating the Cartesian coordinates for all atoms within the molecule requires the

initial determination of the first three atoms as reference points, and then iteratively calculating the position of the next atom with respect to the previous three atoms to which it is bonded. This process includes rotational and translational transformations of the Cartesian coordinates of the previous three atoms, aligning them with the origin, in order to simplify the geometry of the problem. The Cartesian coordinates corresponding to the atoms listed in the above example are listed as follows in what is called a PDB file format. Observe that the first bond is placed along the z axis and the bond angle formed by the first three atoms is placed in the xz plane. Also note the atom types have changed to a more explicit notation which was read in from another file called the RTF file.

	Atom Type	x	y	z
ATOM	NT	0.000	0.000	0.000
ATOM	CT	0.000	0.000	1.453
ATOM	CT	1.457	0.000	1.915
ATOM	C6R	1.773	1.208	2.746
ATOM	C6R	2.233	2.367	2.139

Example of PDB File Format

The calculation of the Cartesian coordinates of all atoms within the molecule is necessary for every evaluation of the energy function corresponding to a particular conformation. The Cartesian coordinates are necessary to calculate the atomic distances between atoms for the non-bonded term of the energy function, as well as to calculate dependent bond angles and dihedral angles which are not explicitly represented in the Z-matrix.

Using the systems approach to design the data structure for the molecular energy model, it follows that two major categories of information must be represented. The input to the model consists of both structural information to represent the three dimensional conformation of the molecule, as well as parameter information required to calculate the components of the energy function. The required output is the Cartesian coordinates of each of the atoms of the molecule.

The first requirement is to design a data structure which retains the structural information present in the Z-matrix. Additionally, the data structure must include the Cartesian coordinates of each atom. The energy function requires access to constant parameters for each bonded term of the energy function. These parameters are determined by the specific combination of atom types in the bonded relationship. The number of bonded relationships which exist in a molecule includes those explicitly represented in the Z-matrix, as well as those bonded relationships which can be implicitly determined from the Z-matrix. The number of bonded relationships is roughly of order n, where n is the number of atoms present in the molecule. The number of non-bonded atom pairs is approximately $n(n-1)/2$. The number of atoms in the molecule is known a priori; however, the number of bonded and non-bonded interaction terms depend upon the structural information present in the Z-matrix. Consequently, a two level approach to the data structure design was implemented. The first level of information pertained to the specific information corresponding to each individual atom. The next level of information pertained to the specific information required to evaluate each component of the energy function and was generated from the first level of information. This approach resulted in two distinct data structures:

```
typedef struct
{
   char type_of_atom[ATOM_NAME_LENGTH]; /* atom type - C, N, etc */
   double charge;          /* atomic charge of atom              */
   double x, y, z;         /* Cartesian coordinates of atom      */
   int atom_j;             /* atom number of direct parents      */
   double bond_length;     /* fixed bond length between i & j    */
   int bond_group;         /* 0=Fixed, 1=Independent             */
   int atom_k;             /* third atom defining bond angle     */
   double bond_angle;      /* fixed bond angle between i, j & k  */
   int angle_group;        /* 0=Fixed, 1=Independent             */
   int atom_l;             /* fourth atom forming dihedral       */
   double dihedral_angle;  /* independent dihedral angle         */
   int dihedral_group;     /* 0=Fixed, 1=Independent             */
} ATOM_TYPE;
```

```
typedef struct group
{
   int i;
   int j;
   int k;
   int l;
   double k_parm;        /* constant parameter */
   double bodanglephi;   /* bond, angle, or phase parameter */
   int n_or_phase;       /* n or phase parameter */
   struct group *next
} GROUP_LIST;
```

The first data structure resembles the contents of the Z-matrix, with the addition of fields to hold the Cartesian coordinates of each atom. The entire molecule is represented as an array of records of type ATOM_TYPE. Any atom specific information can then be referenced by atom number and field, as in atom[i].field. The second data structure is a dynamic data structure which contains the atom numbers and constant parameters associated with a bonded or non-bonded interaction term of the energy function. The linked list structure can contain as many records as there are bonded and non-bonded interactions. Separate lists were created for each component of the energy function, as well as whether the structural geometry of the bond relationship is fixed, dependent, or independently variable. This resulted in ten separate linked lists, which were instantiated as follows:

```
GROUP_LIST *Non_bond;
GROUP_LIST *Fixed_bond;
GROUP_LIST *Depend_bond;
GROUP_LIST *Indep_bond;
GROUP_LIST *Fixed_angle;
GROUP_LIST *Depend_angle;
GROUP_LIST *Indep_angle;
GROUP_LIST *Fixed_dihedral;
GROUP_LIST *Depend_dihedral;
GROUP_LIST *Indep_dihedral;
```

The first step in initializing the data structures begins by reading in the Z-matrix as an array of atoms of type ATOM_TYPE. After reading the Z-matrix into the appropriate fields of the array of records, a function called create_groups is executed which builds the

lists of atoms with bonded and non-bonded interactions between them. This is done by systematically comparing the indices of atoms to which each atom is bonded. All bonded relationships identified in the Z-matrix are either independently variable or fixed, depending on the flag field for each variable. All bonded relationships which exist, but are not explicitly represented in the Z-matrix are identified and added to the dependent variable lists and there values are determined by the values of the dependent and independent variables. Non-bonded atom pairs are those pairs of atoms which are not involved in a bond, bond angle, or dihedral relationship with each other. After each of the ten group lists are created, the parameters associated with the atom types and the term of the energy function, are found in a parameter file and stored in the list structure. This allows the parameters from the parameter file to be read only once, and makes all required parameters for each evaluation of the energy function available from this point forward.

4.1.2 Parameter Specifications File. The constant parameters associated with the various components of the energy function are constants which have been experimentally derived from actual observed data. The source of parameters for this research is the parameter file used by another software package called CHARMm, specifically version 22.0, last updated 92/10/05. This file contains the constant parameters associated with bond lengths, bond angles, dihedral angles, and non-bonded pairs. The format of the file varies according to the number of atoms involved in the interaction; however, the general format is as follows.

[atom_type 1] [atom_type 2] [atom_type 3] [atom_type 4] [1st parameter] [2nd parameter]

Parameter File Format

The third and fourth atom types are only present for bond angle and dihedral angle parameters, respectively. The parameter file is organized into subsections, with headings

separating the different types of parameters according to the term of the energy function to which they corresponded. The order of the groups starts with the parameters for the bond energy term of the energy function. There are two empirical parameters for bond energy; the leading constant term, and equilibrium radius, both of which depend upon the types of atoms in the bonded pair. The next set of parameters in the parameter file are those associated with bond angle energy. Again there are two parameters; the first is the leading constant term, and the second is the equilibrium bond angle. The third set of parameters are those associated with the dihedral angle energy term of the energy function, and include a leading constant term and an equilibrium dihedral angle. The parameters for bond and dihedral angles depend upon the types and the order of the three and four atoms, respectively, involved in the interaction. Because of the combinatoric nature of ordered relations, many of the dihedral parameters are only specified by specific second and third atom types, in order to reduce the size of the parameter file. Generic atom type positions are identified in the parameter file with a single capital "X" placed in the field normally used for the atom type.

The next set of parameters listed in the parameter file, after the dihedral parameters, are parameters associated with improper angle energy. The improper angle energy is another component of the energy function which can be modeled empirically; however, this term is not included in this research implementation because of its relatively small contribution, and also to facilitate comparisons to other published results that did not model the improper angle energy term. Consequently, this section of the parameter file is skipped to reach the final set of parameters used; the parameters associated with the non-bonded interaction term of the energy function. Unlike the other parameters so far, which correspond directly to terms within the energy function, the non-bonded section of the parameter file provides the values for E_{min} and R_{min} for each individual atom type. The corresponding parameter values for the non-bonded term of the energy function are

derived from the following relationships, where i and j are the numbers corresponding to the pair of atoms involved in the non-bonded interaction.

$$E_{ij} = E_{\min_i} + E_{\min_j}$$
$$R_{ij} = R_{\min_i} + R_{\min_j}$$
$$A_{ij} = R_{ij}(E_{ij})^{\frac{1}{12}}$$
$$B_{ij} = R_{ij}(-2E_{ij})^{\frac{1}{6}}$$

Locating every required parameter by directly searching the parameter file would require an extensive amount of I/O time. Consequently, in order to minimize the amount of I/O used to read the data from the parameter file, the first set of parameters is read into an array structure, which is then used during the search process to locate the appropriate parameters according to the atom types present. After all the required parameters associated with that section of the parameter file are located and stored in the group list structure, the array of parameters is cleared and the next set of parameters are read in from the parameter file. This process continues until all parameters required to evaluate the energy function are stored within the group list structure; thus allowing the parameter file to be closed since it is no longer needed.

4.1.3 Implementation of Empirical Energy Equation. The implementation of the empirical energy function follows a functional programming paradigm where each component of the energy function is implemented as a separate function. This facilitates a summation operation which is performed for each of the individual groups lists corresponding to the separate terms of the energy function, as well as allowing analysis of the individual contribution of each component of the energy function. Since all the parameters for each of the energy terms are stored in the group list structure, the

functional implementation of each term of the energy function can be coded so that the constants and variables are passed as parameters to the function. The following pseudo code demonstrates the functional programming format used for the individual terms of the energy function.

```
energy_component(K1, K2, x)
{
     contribution = f(K1, K2, x)
     return(contribution)
}
```

The constants K1 and K2 correspond to the leading coefficient and the equilibrium constant found in the bond, bond angle, and dihedral angle energy terms of the energy function. The individual contribution of this bonded interaction is equal to a function of K1, K2, and the variable x, which is either the bond length, bond angle or the dihedral angle corresponding to the geometry of the group of atoms in the specific conformation being analyzed.

The summation of the total energy is accomplished by indexing through each of the group lists, calling the appropriate energy component function and summing over all terms of the energy function. The following pseudo code demonstrates the summation of the total energy over each component of the energy function.

```
total_energy = 0.0
for i = 1 to number_of_group_lists
{
   indexPtr = group_list[i]
   energy_contribution = 0.0
   while (indexPtr ≠ NULL)
   {
      energy_contribution = energy_contribution +
         energy_component(indexPtr->K1, indexPtr->K2, indexPtr->x)
      indexPtr = indexPtr->next
   }
   total_energy = total_energy + energy_contribution
}
```

Although the above pseudo code does demonstrate the general process of how the energy function is evaluated, the actual implementation differs for several reasons. First, the function called to evaluate the individual energy component depends upon the type of group list being evaluated, since each group list is associated with a specific term of the energy function. Second, the source of the variable x, also depends upon the type of group list. Fixed variable group lists retrieve the variable x from the original data stored in the array of records structure initialized from the Z-matrix. Independent variable group lists acquire a candidate solution for x from the encoded string representation of the GA. Finally, dependent variable group lists must call an intermediate function to determine the variable x from the Cartesian coordinates of the individual atoms. Similarly, the evaluation of the non-bonded energy requires the intermediate calculation of the distance between two non-bonded atoms, in order to calculate the individual energy contribution of the pair of atoms.

4.1.4 Validation of the Energy Model. The process used to validate the accuracy of this implementation of the energy model involved calculating the initial energy of a test molecule, and comparing this energy to that which is obtained using the CHARMm software package for the same molecule. Both models used the same Z-matrix from which to derive the geometric information relating to the specific conformation of the test molecule. The energy model used by CHARMm is also empirically based, using the same parameter file and the same equations for the individual energy components. The CHARMm model, however, does model other energy components which are not included in the GA model, such as the improper angle energy. CHARMm does have the capability to report the individual contributions of each term of the energy function, consequently allowing a direct comparison between the energies found with the GA energy model and the energies determined by CHARMm. Since the CHARMm software package is widely accepted as an accurate model of molecular energy, the GA model was considered to be

sufficiently validated when its calculated energy contributions agreed with those reported by CHARMm for the same molecule.

4.2 Integration of the Energy Model into the Genetic Algorithm

The energy model serves as the GA's fitness function to evaluate the fitness of population members, thus determining their survival rate into the next generation. The interface between the GA and the energy model is through the function called eval, which evaluates the fitness of each member of the population. Considering the energy for each population member must be calculated for each generation, several steps were taken to maximize the efficiency of the energy calculation. Finally, the clean interface between the GA and the energy model allowed the integration of the model into several different versions of the GA.

4.2.1 Evaluation of Fitness. With each generation, the fitness of each population member is determined by calling the eval function. The call to the function eval passes a population string as its argument. It is the encoding of the GA string which provides the interface between the energy model and the GA, since each string corresponds to a unique spatial conformation of the molecule being modeled. The string representation is decoded into the independent variables within the molecule, such as bond lengths, bond angles, or dihedral angles. After all independent variables are decoded, and in conjunction with the structural information retained from the Z-matrix, the Cartesian coordinates of each atom can be determined. The calculation of the molecular energy follows by indexing through the previously defined group lists, summing up all of the individual components of energy to arrive at the total energy. This total energy corresponds to the specific conformation defined by the string representation and is returned as a value of fitness.

The GA uses the returned energy value directly to proportionally allocate strings into the next generation. In order to evolve solutions which minimize the energy of the

molecule, the GA must allocate more copies in future generations to solutions with high fitness levels (low energy values). With each successive generation, the solutions correspond to three dimensional conformations with lower and lower energies. After a predefined number of fitness evaluations, the best solution found so far is taken as the final solution, and execution is halted. To evaluate the quality of the final solution, the corresponding Cartesian coordinates of each of the atoms are printed to a file in a particular format which can then be read by the QUANTA software package which renders an image of the molecule on the screen for examination.

4.2.2 Efficiency Considerations. Since the execution of the genetic algorithm is dominated be the time required to evaluate the fitness of each string for every generation, every consideration should be given to maximizing the efficiency of the calculation of the molecular energy. One way to accomplish this would be to eliminate any unnecessarily redundant calculations. For most energy minimization problems, only a subset of the dihedral angles are allowed to vary independently. The bond lengths, bond angles, and the remaining dihedral angles are held fixed. Consequently, the energy associated with these fixed interactions can be calculated once and stored, rather than recalculating it for every evaluation. This was accomplished by implementing a global variable called fixed_energy, which stores the sum of all the fixed energies calculated during initialization. The total energy is for each evaluation starts with the fixed energy, and then proceeds to add the contributions from the independently variable energy terms, the dependently variable terms, and the non-bonded interaction term which is dependent on the global structure.

Another efficiency consideration is the availability of all the parameters necessary to evaluate the empirical energy function. In order to avoid searching for the parameters with each evaluation, all necessary parameters are identified during the initialization phase and stored in a record structure of the atom group lists. The parameter file is only searched once, and all subsequent references to parameters can be made directly.

Consequently, the evaluation of the total energy can proceed by stepping through each atom group list, and calling the appropriate energy component function with the parameters passed as arguments.

V. Results and Evaluation

In order to evaluate the performance of GA's as a conformational analysis tool, the developed potential energy model is integrated into both a sequential and a parallel implementation of a simple genetic algorithm based on GENESIS (Grefenstette, 1986), and tested on a real protein. The purpose of the experiment was two fold: first to determine whether or not GA's are an effective search technique for minimizing molecular energies for conformational analysis of proteins, and also to assess the scalability of GA's on parallel computer architectures. To accomplish the first objective, the GA is shown to effectively minimize the energy of the test problem. The second objective is accomplished by demonstrating near linear speedup of a parallel implementation of the GA with no corresponding degradation in solution quality.

5.1 Sequential Implementation Test Results

5.1.1 Development of Test Set. The test problem chosen is the molecule [Met]-Enkephalin, which is a five residue peptide. This molecule was chosen both for its small size and because it has a known conformational structure, including a beta turn. The bond lengths and bond angles were held fixed, while the dihedral angles along the backbone (four each of ϕ, ψ, and ω angles) and 9 sidechain dihedral angles (χ angles) of the protein where allowed to vary independently, for a total of 21 independent variables. A resolution of approximately 0.35 degree increments was achieved by encoding 10 binary characters for each independent variable that resulted in a total string length of 210 bits for each member of the population of strings.

5.1.2 Experimental Procedure. The sequential implementation is performed on a Sparc Station II, executing in the background. Because of the long execution times, up to 14 hours per run, the number of experiments performed was limited to ten individual runs. Various population sizes and mutation rates were performed in order to gain empirical support for choosing appropriate GA parameters. The best run was evaluated on the basis of final solution quality and also upon the population convergence evidenced by the degradation in population variance.

The effect of population size on convergence is demonstrated in Figures 5-8. With a relatively small population size of 50, the GA converges to a single solution after 25,000 fitness evaluations. The convergence is more evident in Figures 6 and 7 where the variance is shown to decrease toward zero and the average population fitness becomes equal to the best fitness found. Increasing the population size provides more genetic material, allowing the GA to continue to explore the conformational search space. This continued exploration is demonstrated in Figure 8, where the population does not converge but rather the algorithm stops when reaching the predefined maximum number of 500,000 fitness evaluations.

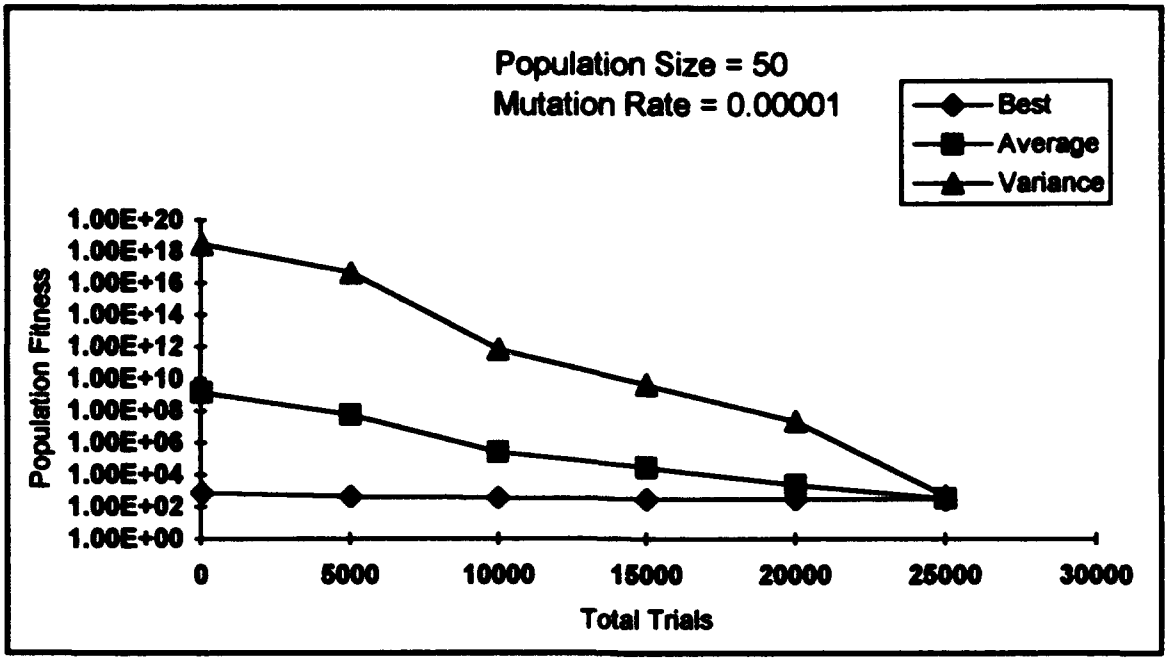

Figure 5. Convergence After 25,000 Evaluations

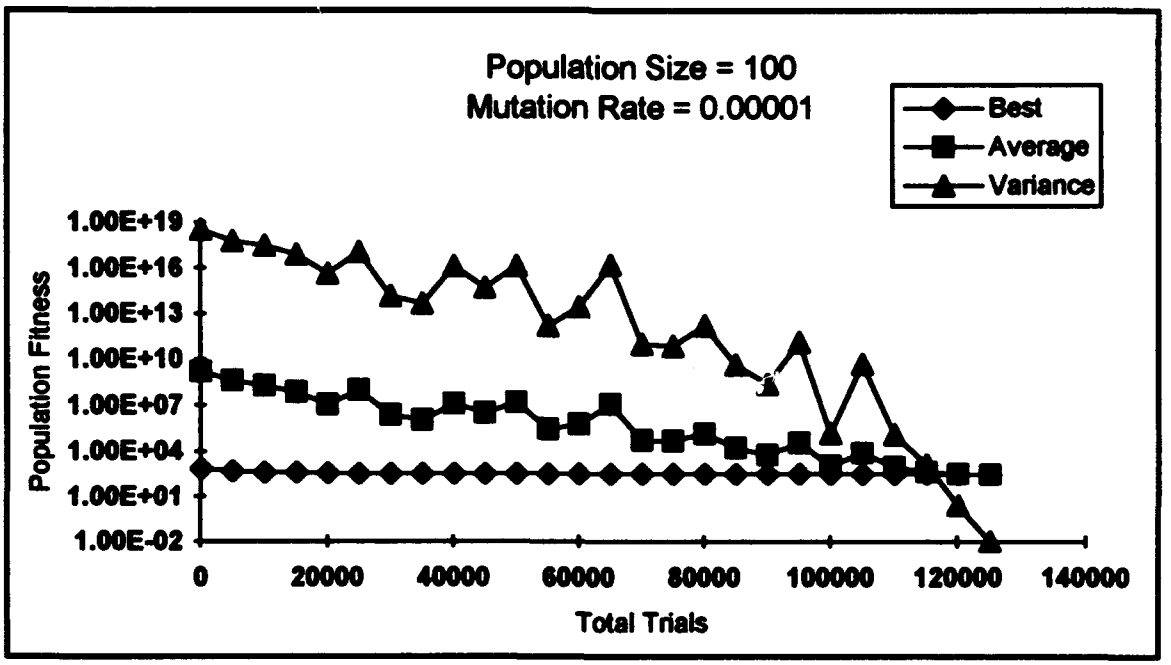

Figure 6. Convergence After 120,000 Evaluations

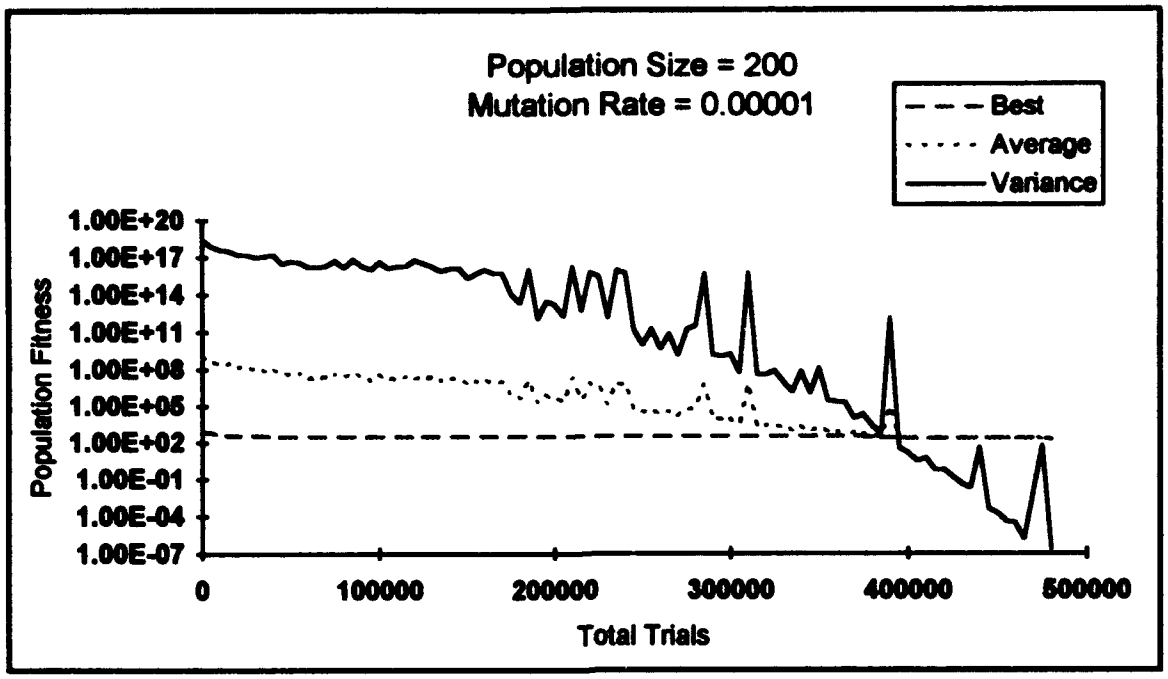

Figure 7. Convergence Just Before Reaching 500,000 Evaluations

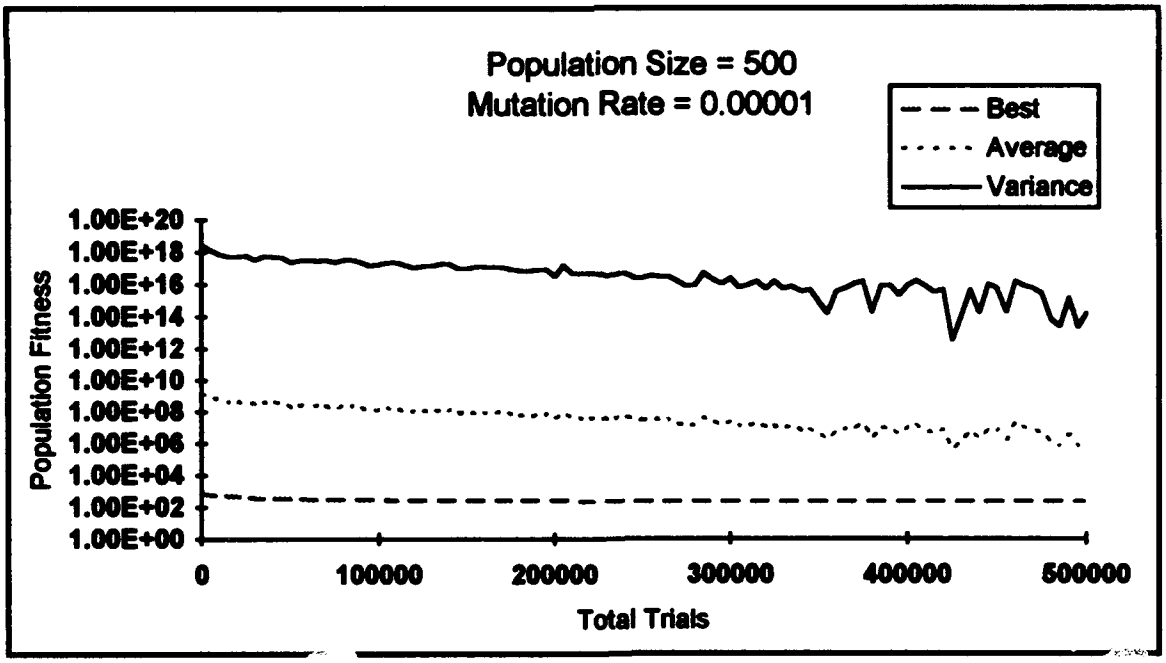

Figure 8. Maintains Population Diversity Till Reaching the Stopping Criteria

The effectiveness of the GA conformational search in terms of solution quality can be seen in Figure 9, which plots the best solution found so far during execution for each of the various population sizes. It is interesting to note that the results from the population size of 200 outperforms the results achieved using a larger population size of 500. It is possible, however, that the population of 500 would have been able to find a better solution, had it been allowed to continue to evolve to the point of convergence. The amount of time allowed for a GA to execute, represents the tradeoff between exploration and exploitation.

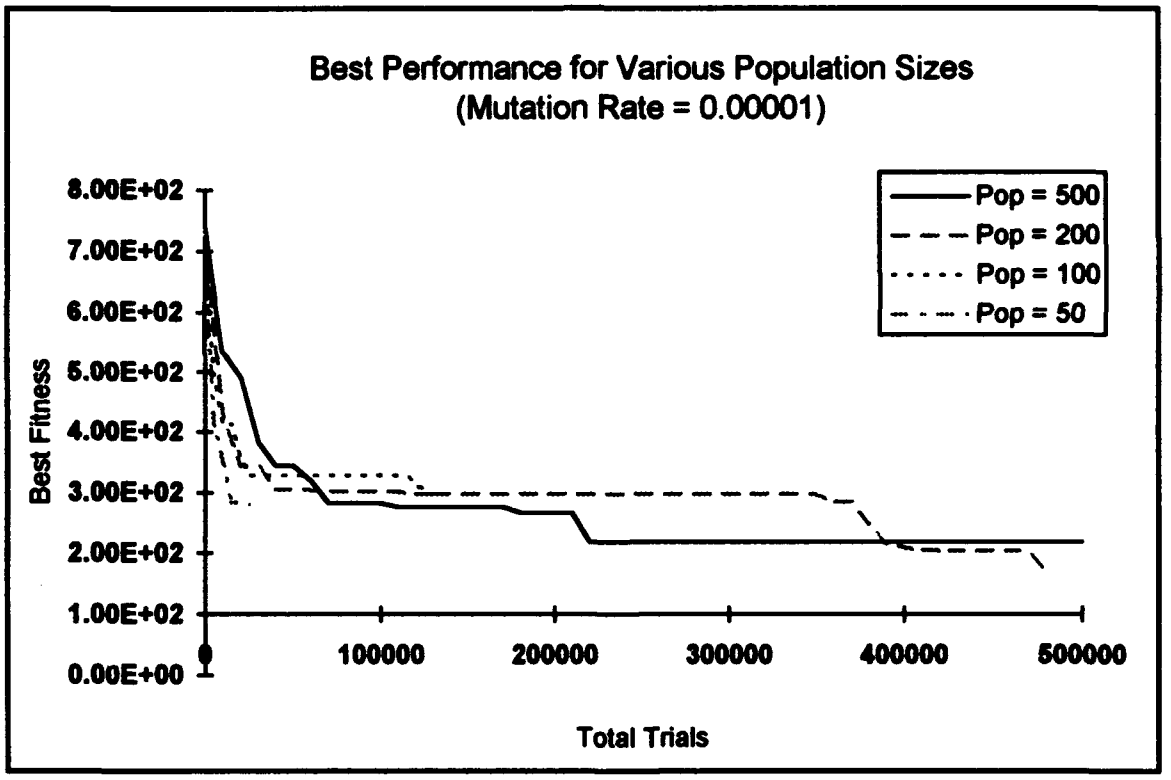

Figure 9. Relative Performance of Various Population Sizes

5.1.3 Evaluation of Performance. For the sequential implementation, the resulting conformation could be directly compared to the accepted known conformational structure. The comparison is performed by comparing the corresponding backbone and

Structure	Energies					
	Bon	Ang.	Dih.	LJ	Elec.	Total
Initial	42.5	5.2	13.8	71.0	-35.5	97
Initial Minimized	0.4	2.4	6.2	-10.4	-34.7	-36
Best GA	0.4	2.6	6.1	-11.6	-36.7	-39

Table 1. Potential Energy Values

sidechain dihedral angles. The lowest energy structure generated from the GA application has been further minimized, using a steepest descent conjugate gradient local minimizer. This resulting structure is then compared to the global energy minimum structure of [Met]-Enkephalin by direct comparison of the backbone and χ dihedral angles.

Table 1 presents the energy values (kcal/mol) for the initial structure, the initial minimized structure, and the best GA result (after minimization). Table 2 provides a comparison between the various backbone dihedral angles (degrees) for the same three methods, plus the results reported by Nayeem, Vila, and Scheraga, (1991).

It is interesting to point out that although the energy has not changed substantially for using the GA compared to locally minimizing the initial structure, the conformational space has been searched. Indeed, for the locally minimized structure the ϕ, ψ angles remain about 180 deg (ϕ: mean = -164; std = 32; ψ: mean = 170; std = 11), while for GA structure the beta-bend is, in part, being reproduced. These results are still being refined, and also note that in Nayeem's (1991) article a relatively wide range of results has been previously reported.

5.2 Parallel Implementation Test Results

5.2.1 Development of Test Problem. Similar to the test problem chosen for the sequential implementation, the molecule [Met]-Enkephalin, was again chosen for its small size and known conformational structure. However, note that in this case the non-polar hydrogen atoms were not explicitly included. All bond lengths and bond angles were

again held fixed, while the same dihedral angles along the backbone and sidechain dihedral angles of the protein were allowed to vary independently. The same angle resolution was also used, resulting in the same total string length of 210 bits for each member of the population of strings.

Residue	Method	Dihedral Angle						
		ϕ	ψ	ω	χ^1	χ^2	χ^3	χ^4
Tyr	Initial	180	180	180	152	120	90	
Tyr	Minimized	169	-164	-179	174	62	87	
Tyr	GA	-59	-156	-179	-179	64	89	
Tyr	Nayeem	-86	156	-177	-173	79	-166	
Gly	Initial	180	180	180				
Gly	Minimized	178	-134	180				
Gly	GA	-173	82	180				
Gly	Nayeem	-154	83	169				
Gly	Initial	180	180	180				
Gly	Minimized	-177	176	-179				
Gly	GA	-87	84	179				
Gly	Nayeem	84	-74	-170				
Phe	Initial	180	180	180	-58	120		
Phe	Minimized	159	-133	178	-80	177		
Phe	GA	163	-134	-176	-75	177		
Phe	Nayeem	-137	19	-174	59	-85		
Met	Initial	180	180	180	-58	180	180	180
Met	Minimized	160	160	180	79	176	-175	180
Met	GA	-158	-158	180	80	-177	176	179
Met	Nayeem	-164	160	-180	53	175	-180	-59

Table 2. Comparison of Dihedral Angles for Various Methods

The lowest energy structure generated from the GA application has been further minimized, using a steepest descent conjugate gradient local minimizer. This resulting structure is then compared to the global energy minimum structure of [Met]-Enkephalin by direct comparison of the backbone and χ dihedral angles.

5.2.2 Experimental Procedure. Preliminary testing was performed on both a serial and a parallel implementation of the GA in order to gain insight into the proper GA settings for mutation rates and population sizes for this particular application. The mutation operator was implemented as a bitwise probability; therefore, in order to achieve a desired probability of changing a string structure, the bitwise mutation rate must determined by the following relationship. The bitwise mutation rate is designated p_m and the probability of mutation for a string structure is designated as P_M, where the length of the string is l. The mutation rate chosen for this experiment was 0.00001, which corresponds to a 0.21% probability of mutating a string structure.

$$P_M = 1 - (1 - p_m)^l$$

The GA was executed on an Intel iPSC/860 with 64 processor elements. In order to be able to scale up to a total of 32 processors, a total population size of 640 was chosen such that the subpopulation size on each node would never become less than 20 individuals per processor. This ensured that there would be sufficient genetic material to perform useful search from local crossover and mutation. In order to evaluate the scalability of the GA, the test problem was executed on various partition sizes ranging from 1 to 32 nodes, by powers of 2.

5.2.3 Evaluation of Performance. Both execution time and solution quality were averaged over 5 runs for each partition size. The results show that the GA achieved near linear speedup as the number of processors was increased (Figure 10). In addition, there was no corresponding loss of solution quality as the subpopulation size on each node was reduced by a factor of n/p, where n is the total population size and p is the number of processors (Figure 11).

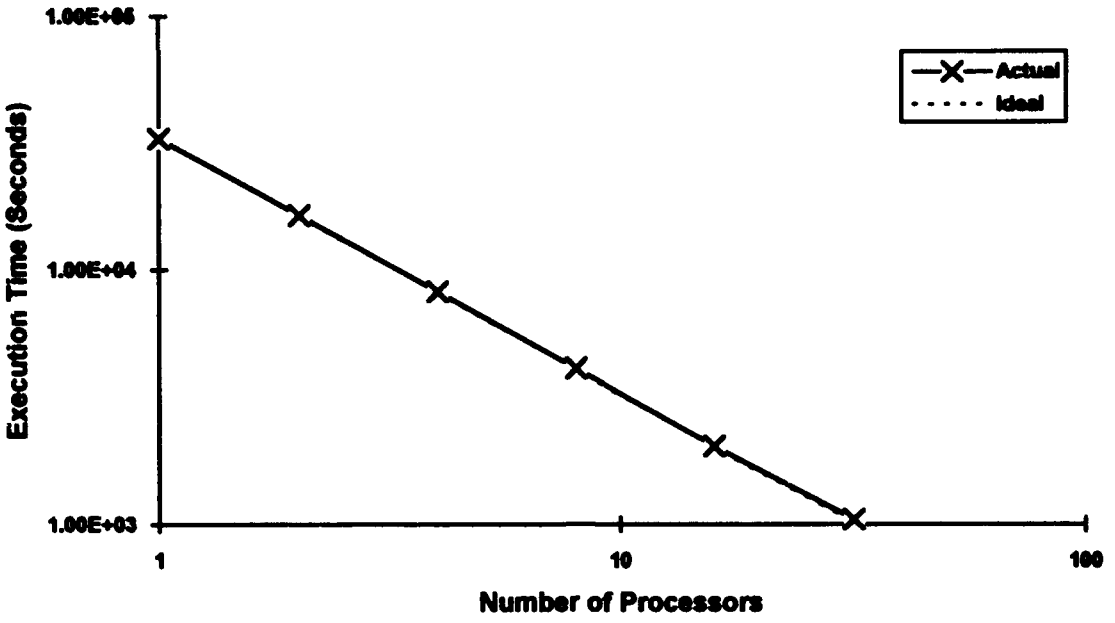

Figure 10. Near Linear Speedup

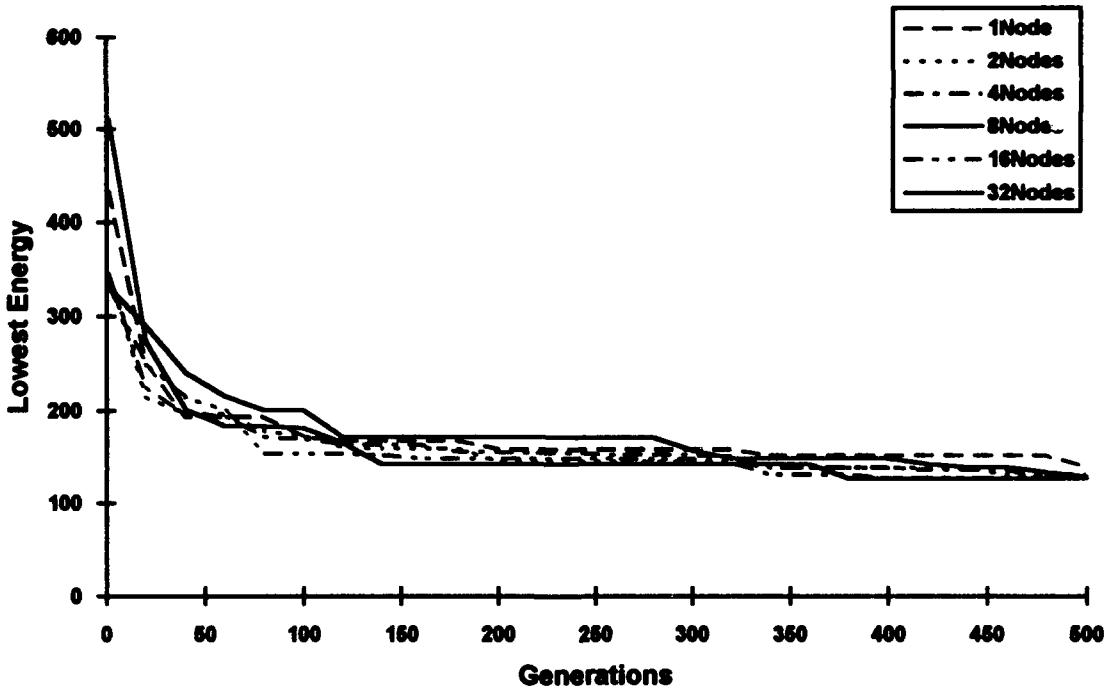

Figure 11. Relative Performance of Parallel GA

VI. Conclusions and Future Research

The genetic algorithm offers many potential advantages over other existing conformational analysis techniques. It has been demonstrated that a GA implementation, using an empirical energy model as the fitness function, is able to optimize independently variable dihedral angles to minimize the energy of the peptide [Met]-Enkephalin. Although this test problem is relatively small, optimizing over only 21 independently varying dihedral angles, the GA is expected to scale well to larger problems. Also, conformational analysis on an entire molecule is not always necessary. Often, only small segments of a large protein may be of interest in order to predict local secondary structure formations.

GA's are probabilistic algorithms and thus can not guarantee the globally minimal solution. However, combining the GA with more traditional local optimization techniques may prove to be a useful conformational analysis technique. The GA could be used to explore large search spaces to locate areas of potential interest, and then a simple hill-climbing algorithm could be used to perform local optimizations in those areas. This approach holds particular promise in the application of conformational analysis, where the GA could be used to locate a population of good candidate solutions which biochemists could then use as a starting point to apply more domain specific knowledge in order to optimize the candidate solutions.

One potentially limiting factor to the scalability of a GA applied to the protein folding problem is the minimum number of population members necessary on each processor in order to perform useful search through crossover. The results reported previously scale very well up to 32 processors; however, increasing the number of

processors would reduce the subpopulation size and eventually may lead to problems of premature convergence as the subpopulation size decreases to an insufficient level. Consequently, as larger and larger parallel platforms become available, the question arises as to what is a sufficient subpopulation size for performing local crossover. This question should be addressed through additional research with larger proteins and larger platforms. Since the evaluation function is of $O(n^2)$, where n is the number of atoms in the molecule, it may be worthwhile to investigate the use of a combined approach of local and global crossover in order to scale up to a larger numbers of processors without correspondingly increasing the total population size.

Guidance for future efforts should emphasize:

- Dynamically controlled parameters
- Improved parallel communications strategies
- Optimization of code for evaluation function
- Scale algorithm to larger parallel platforms
- Scale application to larger problem sizes

Future efforts should be directed towards improving the effectiveness of the GA as a function optimizer. The performance of GAs has been shown to be sensitive to parameter settings. The implementation of dynamically controlled genetic operators offer the potential of fine tuning the parameter settings of the GA during execution rather than being held fixed. Continued efforts to optimize the code should be made in order to reduce execution time. Refinements to the parallel decomposition and communications strategies used in the parallel implementation of the genetic algorithm should also decrease execution time, allowing the GA to be applied to larger problem sizes. Other parallel decomposition strategies should be investigated when the number of parallel processors approaches the total population size. It may become necessary to develop a hybrid type

island model where a subpopulation exists on a partition of the processors rather than on an individual processor.

Bibliography

Aizawa, A. N. & Wah, B. W. (1993). "Dynamic Control of Genetic Algorithms in a Noisy Environment." *Proceedings of the Fifth International Conference on Genetic Algorithms*. 48-55. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Atmar, W. (1990). "Natural processes which accelerate the evolutionary search." *IEEE-ACSSC 90*, 1030-5.

Bäck, T. (1993). "Optimal Mutation Rates in Genetic Search." *Proceedings of the Fifth International Conference on Genetic Algorithms*. 2-8. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Bairoch, A. & Boeckmann, B. (1991). *The SwissProt protein sequence data bank*. Nucleic Acid Research, 19:2247-2249.

Baker, J. E. (1985). "Adaptive Selection Methods for Genetic Algorithms." *Proceedings of the First International Conference on Genetic Algorithms*. 101-11. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Bala, J. & De Jong, K. (1990). "Generation of Feature Detectors for Texture Discrimination by Genetic Search." *IEEE-CH2915-7/90*, 812-8.

Baluja, S. (1993). "Structure and Performance of Fine-Grain Parallelism in Genetic Search." *Proceedings of the Fifth International Conference on Genetic Algorithms*. 155-62. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Bhattacharjya, A., Becker, D., & Roysam, B. (1992). "A genetic algorithm for intelligent imaging from quantum-limited data." *Signal Processing* 28, 335-48.

Bilbro, G. & Snyder, W. (1991). "Optimization of Functions with Many Minima." *IEEE Trans. Systems, Man & Cybernetics*, 21, No. 4, Jul/Aug 1991, 840-9.

Bishop, R. (1990). "Identifying Induction Machine Parameters Using A Genetic Optimization Algorithm." *IEEE Proceedings - 1990 Southeastcon*, 476-9.

Blanton, J. & Wainwright, R. (1993). "Multiple Vehicle Routing with Time and Capacity Constraints using Genetic Algorithms." *Proceedings of the Fifth International Conference on Genetic Algorithms*. 452-9. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Blommers, M. J., Lucasius, C. B., Kateman, G., and Kaptein, R. (1992). "Conformational Analysis of a Dinucleotide Photodimer with the Aid of the Genetic Algorithm." *Biopolymers*. 32:45-52.

Booker, L. B. (1985). "Improving the Performance of Genetic Algorithms in Classifier Systems." *Proceedings of the First International Conference on Genetic Algorithms*. 80-92. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Bramlette, M. F. (1991). "Initialization, Mutation and Selection Methods in Genetic Algorithms for Function Optimization." *Proceedings of the Fourth International Conference on Genetic Algorithms*. 100-7. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Brassard, G. & Bratley, P. (1988). *Algorithmics, Theory and Practice*. Englewood Cliffs, New Jersey: Prentice Hall.

Brinkman, D. J., Merkle, L. D., Lamont, G. L., & Pachter, R. (1993). "Parallel Genetic Algorithms and Their Application to the Protein Folding Problem." Proceedings of the 1993 Intel Supercomputer Users Group Meeting.

Buckles, B. P., Petry, F. E., & Kuester, R. L. (1990). "Schema Survival Rates and Heuristic Search in Genetic Algorithms." *International Conference on Tools for Artificial Intelligence*, IEEE-TAI 90, 322-7.

Cartwright, H. M. & Mott, G. F. (1991). "Looking Around: Using Clues from the DATA Space to Guide Genetic Algorithm Searches." *Proceedings of the Fourth International Conference on Genetic Algorithms*. 108-14. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Chan, H. & Dill, K. (1993). "The Protein Folding Problem." *Physics Today*, February, 24-32.

Chandy, K. M. & Misra, J. (1989). *Parallel Program Design, A Foundation*. Reading, Massachusetts: Addison-Wesley Publishing Company.

CHARMm (1992). Parameter File for CHARMm version 22.0, Copyright 1992, Molecular Simulations Incorporated.

Cheung, J. & Chang, C. (1991). "Expert Search in Neural Network Optimization." IEEE-CH2819-1/90, 29-32.

Corcoran, A. (1993). "Reducing Disruption of Superior Building Blocks in Genetic Algorithms." Submitted to *International Conference on Genetic Algorithms - 1993*.

Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1990). *Introduction to Algorithms.* Cambridge, Massachusetts: The MIT Press & McGraw-Hill Book Company.

Davidor, Y., Yamada, T., & Nakano, R. (1993). "The ECOlogical Framework II: Improving GA Performance At Virtually Zero Cost." *Proceedings of the Fifth International Conference on Genetic Algorithms.* 171-6. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Davis, L. (1989). "Adapting Operator Probabilities In Genetic Algorithms." *Proceedings of the Third International Conference on Genetic Algorithms.* 61-7. San Mateo CA: Morgan Kaufmann Publishers, Inc.

de la Maza, M. & Tidor, B. (1993). "An Analysis of Selection Procedures with Particular Attention Paid to Proportional and Boltzmann Selection." *Proceedings of the Fifth International Conference on Genetic Algorithms.* 124-31. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Deb, K. & Goldberg, D. E. (1989). "An Investigation of Niche and Species Formation in Genetic Function Optimization." *Proceedings of the Third International Conference on Genetic Algorithms.* 42-50. San Mateo CA: Morgan Kaufmann Publishers, Inc.

DeCegama, A. L. (1989). *The Technology of Parallel Processing, Parallel Processing Architectures and VLSI Hardware, Volume I.* Englewood Cliffs, New Jersey: Prentice Hall, Inc.

De Jong, K. & Spears, W. (1993). "On The State of Evolutionary Computation." *Proceedings of the Fifth International Conference on Genetic Algorithms.* 618-23. San Mateo CA: Morgan Kaufmann Publishers, Inc.

De Jong, K. (1975). An analysis of the behavior of a class of genetic adaptive systems. (Doctoral dissertation, University of Michigan). *Dissertation Abstracts International.* 36(10), 5140B.

Dontas, K. & De Jong, K. (1990). "Discovery of Maximal Distance Codes Using Genetic Algorithms." IEEE-CH2915-7/90, 805-11.

Dorigo, M. & Maniezzo, V. (1993). "Parallel Genetic Algorithms: Introduction and Overview of Current Research." *Parallel Genetic Algorithms,* 5-33.

Dymek, A. (1992). *An Examination of Hypercube Implementations of Genetic Algorithms.* MS thesis, AFIT/GCS/ENG/92-M, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, March 1992.

Eshelman, L. J. & Schaffer, J. D. (1993). "Crossover's Niche." *Proceedings of the Fifth International Conference on Genetic Algorithms*. 9-14. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Eshelman, L. J. & Schaffer, J. D. (1991). "Preventing Premature Convergence in Genetic Algorithms by Preventing Incest." *Proceedings of the Fourth International Conference on Genetic Algorithms*. 115-22. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Fogarty, T. (1989). "Varying the Probability of Mutation in the Genetic Algorithm." *Proceedings of the Third International Conference on Genetic Algorithms*. 104-9. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Fogel, D. B. (1990). "Simulated Evolution: A 30-Year Perspective." *IEEE-ACSSC 90*, 1009-14.

Fogel, G. B. (1993). "An Introduction to the Protein Folding Problem and the Potential Application of Evolutionary Programming." *The Second Annual Conference on Evolutionary Programming*. 170-7. San Diego, CA: Evolutionary Programming Society.

Fogel, L. (1990). "The Future of Evolutionary Programming." *IEEE-ACSSC 90*, 1036-8.

Fonseca, C. M. & Fleming, P. J. (1993). "Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization." *Proceedings of the Fifth International Conference on Genetic Algorithms*. 416-23. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Fontain, E. (1992). "Application of Genetic Algorithms in the Field of Constitutional Similarity." *J. Chem. Inf. Comput. Sci.* 32:748-52.

Forrest, S. & Mitchell, M. (1993). "Relative Building-Block Fitness and the Building-Block Hypothesis." in *Foundations of Genetic Algorithms 2*, D. Whitley (ed.), San Mateo CA: Morgan Kaufmann Publishers, Inc.

Goldberg, D. E., Deb, K., Kargupta, H., & Harik, G. (1993). "Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms." *Proceedings of the Fifth International Conference on Genetic Algorithms*. 56-64. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Goldberg, D. E., Deb, K., & Clark, J. H. (1992). "Genetic Algorithms, Noise, and the Sizing of Populations." *Complex Systems*. 6:333-62.

Goldberg, D. E. and others. (1991). "Don't Worry, Be Messy." *Proceedings of the Fourth International Conference on Genetic Algorithms*. 24-30. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Goldberg, D. E., Deb, K., & Korb, B. (1990). "Messy Genetic Algorithms Revisited: Studies in Mixed Size and Scale." *Complex Systems*. 4:415-44.

Goldberg, D. E., Korb, B., & Deb, K. (1989). "Messy Genetic Algorithms: Motivation, Analysis, and First Results." *Complex Systems*. 3:493-530.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading MA: Addison-Wesley Publishing Company.

Goldberg, D. E. & Richardson, J. (1987). "Genetic Algorithms with Sharing for Multimodal Function Optimization" *Proceedings of the Second International Conference on Genetic Algorithms*. 41-9. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Goldberg, D. E. & Segrest, P. (1987). "Finite Markov Chain Analysis of Genetic Algorithms." *Proceedings of the Second International Conference on Genetic Algorithms*. 1-8. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Goldberg, D. E. & Smith, R. (1987). "Nonstationary Function Optimization Using Genetic Algorithms with Dominance and Diploidy." *Proceedings of the Second International Conference on Genetic Algorithms*. 59-68. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Gordon, V. S. & Whitley, D. (1993). "Serial and Parallel Genetic Algorithms as Function Optimizers." *Proceedings of the Fifth International Conference on Genetic Algorithms*. 177-83. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Grefenstette, J. J. (1993). "Deception Considered Harmful." in *Foundations of Genetic Algorithms 2*, D. Whitley (ed.), San Mateo CA: Morgan Kaufmann Publishers, Inc.

Grefenstette, J. J. (1986). *A User's Guide to Genesis*. Technical Report, Nashville, TN: Vanderbilt University.

Handley, S. (1993). "Automated Learning of a Detector for α-Helices in Protein Sequences Via Genetic Programming." *Proceedings of the Fifth International Conference on Genetic Algorithms*. 271-8. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Harvey, I. (1993). "The Puzzle of the Persistent Question Marks: A Case Study of Genetic Drift." *Proceedings of the Fifth International Conference on Genetic Algorithms*. 15-22. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Himler, A. & Wechsler, H. (1992). "Suboptimal MAP estimates using A* and genetic algorithms." *Proc. SPIE - Int. Soc. Opt. Eng (USA)*, vol.1607, 27-37.

Hinton, G. E. & Nowlan, S. J. (1987). "How Learning Can Guide Evolution." *Complex Systems*. 1:495-502.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor MI: The University of Michigan Press.

Homaifar, A., Guan, S., & Liepins, G. E. (1993). "A New Approach on the Traveling Salesman Problem by Genetic Algorithms." *Proceedings of the Fifth International Conference on Genetic Algorithms*. 460-6. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Janikow, C. Z. & Michalewicz, Z. (1991). "An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms." *Proceedings of the Fourth International Conference on Genetic Algorithms*. 31-6. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Janikow, C. Z. & Michalewicz, Z. (1990). "A Specialized Genetic Algorithm for Numerical Optimization Problems." *International Conference on Tools for Artificial Intelligence*, IEEE-TAI 90, 798-804.

Jog, P., Suh, J. Y., & Van Gucht, D. (1989). "The Effects of Population Size, Heuristic Crossover and Local Improvement on a Genetic Algorithm for the Traveling Salesman Problem." *Proceedings of the Third International Conference on Genetic Algorithms*. 110-15. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Kling, R. M. & Banerjee, P. (1990). "Optimization by Simulated Evolution with Applications to Standard Cell Placement." *27th ACM/IEEE Design Automation Conference*, 20-5.

Koza, J. (1990). "Genetically Breeding Populations of Computer Programs to Solve Problems in Artificial Intelligence." *IEEE-CH2915-7/90*, 819-27.

Lee, M. A. & Takagi, H. (1993). "Dynamic Control of Genetic Algorithms using Fuzzy Logic Techniques." *Proceedings of the Fifth International Conference on Genetic Algorithms*. 76-83. San Mateo CA: Morgan Kaufmann Publishers, Inc.

LeGrand, S. M. & Merz, K. M. (1993). "The Application of the Genetic Algorithm to the Minimization of Potential Energy Functions." *Journal of Global Optimization*, 3:49-66.

Lengauer. T. (1993). "Algorithmic Research Problems in Molecular Bioinformatics". *Arbeitspapiere der GMD* 748, May 1993.

Maruyama, T., Hirose, T., & Konagaya, A. (1993). "A Fine-Grained Parallel Genetic Algorithm for Distributed Parallel Systems." *Proceedings of the Fifth International Conference on Genetic Algorithms*. 184-90. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Merkle, L. & Lamont, G. (1993). "Comparison of Parallel Messy Genetic Algorithm Data Distribution Strategies." *Proceedings of the Fifth International Conference on Genetic Algorithms*. 191-8. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Merkle, L. (1992). *Generalization and Parallelization of Messy Genetic Algorithms and Communication in Parallel Genetic Algorithms*. MS thesis, AFIT/GCE/ENG/92-D, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1992.

Moed, M. C., Stewart, C. V., & Kelly, R. B. (1991). "Reducing The Search Time Of A Steady State Genetic Algorithm Using the Immigration Operator." *International Conference on Tools for Artificial Intelligence*, IEEE-TAI 91, 500-1.

Montgomery, D. C. (1976). *Design and Analysis of Experiments, Second Edition.* New York: John Wiley & Sons

Muselli, M. & Ridella, S. (1992). "Global Optimization of Functions with the Interval Genetic Algorithm." *Complex Systems*. 6:193-212.

Nayeem, A., Vila, J., & Scheraga, H. (1991). "A Comparative Study of the Simulated-Annealing and Monte Carlo-with-Minimization Approaches to the Minimum-Energy Structures of Polypeptides: [Met]-Enkephalin." *Journal of Computational Chemistry*, Vol.12, No.5, 594-605.

Otten, R. & van Genneken, L. (1990). "The Complexity of Adaptive Annealing." *IEEE-CH2909-0/90*, 404-7.

Pachter, R., Cooper, T. M., Natarajau, L. V., Crane, R. L. & Adams, W. W. (1992). *Biopolymers*, 32:1129-40.

Pachter, R., Cooper, T. M., Crane, R. L., & Adams, W. W. (1993). "Smart Structures and Materials." *'93 SPIE Proceedings*, 1; and in *Bio/Technology Winter Symposia: Protein Engineering and Beyond*, 3:20.

Palmer, M. E. & Smith, S. J. (1991). "Improved Evolutionary Optimization of Difficult Landscapes: Control of Premature Convergence through Scheduling Sharing." *Complex Systems*. 5:443-458.

Pearl, J. (1984). *Heuristics, Intelligent Search Strategies for Computer Problem Solving*. Reading, Massachusetts: Addison-Wesley Publishing Company

Reeves, C. R. (1993). "Using Genetic Algorithms with Small Populations." *Proceedings of the Fifth International Conference on Genetic Algorithms*. 92-9. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Reynolds, R. G. (1990). "The Control of Genetic Algorithms Using Version Spaces." *IEEE-CH2915-7/90*, 342-8.

Rich, E. & Knight, K. (1991). *Artificial Intelligence*. New York: McGraw-Hill, Inc.

Schaffer, J. D., Caruana, R. A., Eshelman, L. J., & Das, R. (1989). "A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization." *Proceedings of the Third International Conference on Genetic Algorithms*. 51-60. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Schaffer, J. D. & Morishima, A. (1987). "An Adaptive Crossover Distribution Mechanism for Genetic Algorithms." *Proceedings of the Second International Conference on Genetic Algorithms*. 36-40. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Schaffer, J. D. (1985). "Multiple Objective Optimization with Vector Evaluated Genetic Algorithms." *Proceedings of the First International Conference on Genetic Algorithms*. 93-100. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Schultz, A. & Grefenstette, J. (1990). "Improving Tactical Plans with Genetic Algorithms." *IEEE-CH2915-7/90*, 328-33.

Schulze-Kremer, S. (1993). "Genetic Algorithms for Protein Tertiary Structure Prediction." *Parallel Genetic Algorithms*, 129-49.

Sen, S. (1993). "Minimum Cost Set Covering Using Probabilistic Methods." *Association for Computing Machinery-SAC '93*, 157-64.

Shaefer, C. (1987). "The ARGOT Strategy: Adaptive Representation Genetic Optimizer Technique." *Proceedings of the Second International Conference on Genetic Algorithms*. 50-8. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Shonkwiler, R. (1993). "Parallel Genetic Algorithms." *Proceedings of the Fifth International Conference on Genetic Algorithms*. 199-205. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Sirag, D. & Weisser, P. T. (1987). "Toward a Unified Thermodynamic Genetic Operator." *Proceedings of the Second International Conference on Genetic Algorithms*. 116-22. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Spears, W. & De Jong, K. (1990). "Using Genetic Algorithms For Supervised Concept Learning." *IEEE-CH2915-7/90*, 335-41.

Spillman, R. (1993). "Genetic Algorithms." *Dr. Dobb's Journal*, February, 26-30+.

Tate, D. M. & Smith, A. E. (1993). "Expected Allele Coverage and the Role of Mutation in Genetic Algorithms." *Proceedings of the Fifth International Conference on Genetic Algorithms*. 31-7. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Theirens, D. & Goldberg, D. E. (1993). "Mixing in Genetic Algorithms." *Proceedings of the Fifth International Conference on Genetic Algorithms*. 38-45. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Tsutsui, S. & Fujimoto, Y. (1993). "Forking Genetic Algorithm with Blocking and Shrinking Modes (fGA)." *Proceedings of the Fifth International Conference on Genetic Algorithms*. 206-13. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Unger, R. & Moult, J. (1993). "A Genetic Algorithm for 3D Protein Folding Simulations." *Proceedings of the Fifth International Conference on Genetic Algorithms*. 581-8. San Mateo CA: Morgan Kaufmann Publishers, Inc.

U. S. Congress, Office of Technology Assessment. (1988). *Mapping Our Genes—The Genome Projects: How Big, How Fast?* No. OTA-BA-373, U. S. Government Printing Office, Washington, D. C.

Von Freyberg, B. & Braun, W. (1991). "Efficient Search for All Low Energy Conformations of Polypeptides by Monte Carlo Methods." *Journal of Computational Chemistry*, Vol.12, No.9, 1065-76.

Whitley, D. (1989). "The *GENITOR* Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best." *Proceedings of the Third International Conference on Genetic Algorithms*. 116-21. San Mateo CA: Morgan Kaufmann Publishers, Inc.

Whitley, D. (1987). "Using Reproductive Evaluation to Improve Genetic Search and Heuristic Discovery." *Proceedings of the Second International Conference on Genetic Algorithms*. 108-15. San Mateo CA: Morgan Kaufmann Publishers, Inc.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3 REPORT TYPE AND DATES COVERED
	December 1993	Master's Thesis

4. TITLE AND SUBTITLE
Genetic Algorithms and Their Application
To The Protein Folding Problem

5. FUNDING NUMBERS

6. AUTHOR(S)
Donald J. Brinkman

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GCE/ENG/93D-02

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Dr. Ruth Pachter, WL/MLPJ, WPAFB, OH 45433

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The protein folding problem involves the prediction of the secondary and tertiary structure of a molecule given the primary structure. The primary structure defines sequence of amino-acid residues, while the secondary structure describes the local 3-dimensional arrangement of amino-acid residues within the molecule. The relative orientation of the sxecondary structural motifs, namely the tertiary structure, defines the shape of the entire biomolecule. The exact mechanism by which a sequence of amino acids (protein) folds into its 3-dimensional conformation is unknown. Current approaches to the protein folding problem include calculus-based methods, systematic search, model building and symbolic methods, random methods such as Monte Carlo simulation and simulated annealing, distance geometry, and molecular dynamics.

Many of these current approaches search for conformations which minimize the internal energy of the molecule. A genetic algorithm (GA), a stochastic search technique modeled after natural adaptive systems, potentially offers significant speedup over other search algorithms because of its inherent parallelizability. The results of applying a parallel GA to the protein folding problem show significant improvement in execution time when compared to serial implementations of the GA. In addition, the parallel GA demonstrates good scalability characteristics since the communications strategy used to manage the population can be tailored to the parallel architecture.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Genetic algorithm, parallel application, protein folding, energy minimization, conformational analysis	83
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL